A PRIMERA REVISTA DE PROGRAMACION EN CASTELLANO

SOLO PROGRAMADORES

) III, NÚMERO 34

975 Pts.

PROGRAMACIÓN ORIENTADA A OBJETOS

HERENCIA EN VISUAL C++ Y DELPHI

TCL/TK
Widgets con TK

REDES LOCALES
Secuencias de conexión y menús en Novell

WWW
JavaScript
Frames y windows

PACKET DRIVER
Programación

JAVA
Constructores, composición, herencia,...

- Visual Basic Avanzado
- Criptografía
- Creación de una intranet con Linux

CONTENIDO DEL CD-ROM

- Demo Rational Rose 4.0
- Basic 4 Java
- Olectra Chart
- Solid Server 2.2
- PGP 2.6. 3i
- Internet Toolpack 3.01

8 413042 303299 00034

TOWER

SÓLO PROGRAMADORES

Número 34
SÓLO PROGRAMADORES
es una publicación de
TOWER COMMUNICATIONS

Director Editor
Antonio M. Ferrer Abelló
aferrer@towercom.es

Directora Adjunta
Amelia Noguera
anoguera@towercom.es

Colaboradores
Enrique Díaz, Arsenio Molinero, Carlos Álvaro, Enrique de la Lastra, Francisco José Abad Cerdá, Jose Antonio Collar, Javier Sarsa, Oscar Prieto Blanco, Ernesto Schmitz, Alejandro Reyero, Emilio Postigo Rian, Chema Álvarez, J. Antonio Mendoza, J. Merseguer, José Romero Cuñac.

Jefe de Diseño
Fernando García Santamaría

Maquetación
Clara Francés

Tratamiento de Imagen
María Arce Giménez

Imagen de Portada
Fernando Núñez

Publicidad
Erika de la Riva (Madrid)
Tel.: (91) 661 42 11
eriva@towercom.es

Publicidad
Pepín Gallardo (Barcelona)
Tel.: (93) 213 42 29

Suscripciones
Isabel Bojo
Tel. (91) 661 42 11 Fax: (91) 661 43 86
suscrip@towercom.es

Filmación
Megatipo

Impresión
G. Reunidas

Distribución
SGEL

Director General
Antonio M. Ferrer Abelló

Director Financiero
Francisco García Díaz de Liaño

Director de Producción
Carlos Peropadre

Directora Comercial
Carmina Ferrer
carmina@towercom.es

Director de Distribución
Enrique Cabezas García

Redacción, Publicidad y Administración
C/ Aragoneses, 7
28108 Pol. Ind. Alcobendas (MADRID)
Telf.: (91) 661 42 11 / Fax: (91) 661 43 86

Depósito legal: M-26827-1994
ISSN: 1134-4792

EDITORIAL

Cuando las máquinas ¿piensan?

¿Quién puede decir qué saben las máquinas? Y quién puede afirmar qué es lo que llegarán a saber. Si es que esta palabra "saber", como sinónimo de "conocer", puede ser aplicada a una máquina. El lenguaje C ha sido el utilizado por los programadores de la ahora famosa y victoriosa máquina de IBM que ha conseguido ganar al también famoso y ahora vencido Gary Kásparov; mediante la programación en este conocido lenguaje sobre un sistema operativo AIX 6000, el equipo de desarrolladores que han dotado a Deep Blue de "casi inteligencia" ha conseguido ganar al genio.

Y podemos (y también debemos) preguntarnos si han sido realmente los conocimientos que la base de datos del sistema experto que constituye Deep Blue y los algoritmos que fluyen por sus circuitos, los responsables de la victoria, o si la presión de la propia mente de una persona enfrentada a una máquina que tal vez podría ganar, ha llegado a ser tan alta que ha ocasionado que el casi invencible Kaspárov cometiera esos fallos tan tremendos en las primeras jugadas y consiguiera su derrumbamiento psicológico. Desde una revista técnica como es **Sólo Programadores** felicitamos al equipo de programadores de Deep Blue que ha dado un paso muy importante en el camino hacia el desarrollo de la Inteligencia Artificial. Pero siendo seres humanos tenemos que pensar lo que significaría que las máquinas pudieran superarnos, aunque fuera en un juego con reglas claras y basado en un conocimiento muy concreto.

Dejemos a la propia intuición y a los conocimientos de cada uno la opinión sobre estos temas y pasemos a ser algo más pragmáticos. Y qué mejor forma que proponer desde aquí un campeonato de ajedrez entre programas. Dejemos a las personas que sean quienes piensen cómo programar un "jugador de ajedrez" que sea capaz de vencer a otro de sus mismas características, y veamos cuáles son los resultados de dos programas jugando uno contra el otro. Para ello, cómo no, os pedimos que participéis con nosotros en este campeonato decidiendo si os parece interesante. Ya sabéis que tenéis una dirección de correo electrónico a vuestra disposición: solop@towercom.es, donde dirigir vuestros comentarios, dudas, sugerencias, etc. sobre la revista. Ahora os pedimos que también la utilicéis para hacernos saber si os gustaría participar en el campeonato. Las bases y el premio los conoceréis en el próximo número de Sólo Programadores, aunque podéis informaros directamente en la empresa de distribución y venta de material informático AWS (situada en la calle Serrano Jover, 3, de Madrid) o en el teléfono 91-542 50 87.

Y enhorabuena a los partidarios de utilizar Delphi para sus programas, la versión 3.0 ha salido ya al mercado y presenta novedades interesantes, que os contaremos en profundidad. Tal vez pudiera ser elegido como lenguaje para programar el "jugador de ajedrez" que consiga ganar a todos los demás ¿os animáis?


34



SE PRESENTA EL PENTIUM II

Funcionando a 300, 266 y 233 MHz, el procesador combina las tecnologías del Pro con las MMX

El día 7 de mayo Intel anunciaba de nuevo mejoras dentro del fascinante mundo de los chips, que nos abruma cada año con sus cambios. Ahora me actualizo y dentro de nada me vuelvo a actualizar y así continuamos. Dicen que así es la informática.

El Pentium II que Intel nos presenta esta vez pretende ser una combinación explosiva entre su procesador Pentium Pro y la tecnología MMX, de forma que con un sólo producto se proporciona a los usuarios corporativos la potencia para impulsar la informática de empresa, a los pequeños negocios grandes ventajas y, además, hace gala de ser el procesador más rápido de Intel dentro del segmento de las estaciones de trabajo.

Las mejoras tecnológicas son varias:

1) Doble Bus independiente, heredado del Pentium Pro, que resuelve las limitaciones de ancho de banda de las anteriores generaciones de arquitecturas de procesador. Este doble bus está formado por uno del procesador a la caché L2 y otro del sistema a la memoria principal, que permite transacciones paralelas simultáneas en lugar de las transacciones únicas secuenciales de los procesadores anteriores.

2) La tecnología MMX añadida al Pentium II para mejorar las prestaciones de las aplicaciones de audio, vídeo y gráficos, y para acelerar los procesos de codificación y compresión de datos.

3) La llamada tecnología de Ejecución Dinámica, que permite el procesamiento de un mayor número de datos en paralelo para un periodo temporal dado.

4) El encapsulado *Single Edge Contact* (S.E.C) permite que los componentes se monten en un substrato y se encierren totalmente en un cartucho de metal y plástico, formando así el procesador.

EL núcleo del procesador se basa en la arquitectura P6 de Intel y se fabrica en tecnología de proceso 0,35 micras con 7,5 millones de transistores. Se puede disponer de las frecuencias a 266 y 233 Mhz con caché L2 de 512 Kb.

Los sistemas basados en el procesador Pentium II ofrecen elevadas prestaciones cuando ejecutan distintos sistemas operativos.

Varias empresas del sector han comenzado a comercializar equipos con Pentium II.

Para más información consultar la dirección de correo electrónico:

http:// www.intel.com/pressroom

TANDEM PRESENTA ECLIPSE
El sistema UNIX sirve como base para la aplicación de esta nueva tecnología

Tandem Computers ha presentado una nueva tecnología de clustering, llamada Eclipse, que se basa en el sistema UNIX y que aplica la arquitectura de interconexión ServerNet. La compañía piensa acercar esta tecnología al mercado de telecomunicaciones de sistemas UNIX y licenciar Eclipse junto con la tecnología ServerNet a otros proveedores comerciales de UNIX.

Esta tecnología proporciona una arquitectura de conexión para clustering abierto, entre cualquier sistema, posibilitando así que los sistemas basados en UNIX puedan escalar, creciendo a medida que se necesita.

Como característica de Eclipse destaca su capacidad de imagen de sistema simple (SSI), que permite que el cluster aparezca en el sistema UNIX como un único servidor para las aplicaciones, usuarios y administradores. Además está diseñada para soportar miles de nodos del sistema UNIX que se agrupan y funcionan como un único sistema de gran tamaño, eliminando de esta manera la necesidad de que las aplicaciones, utilidades y administradores sean conscientes del cluster.

JAVA vuelve a ser elegido para desarrollar aplicaciones de importancia

Bentley presenta un Sistema de Información Geográfica (GIS) que se basa en Java/CORBA y presenta la aplicación a la OGC (Open GIS Consortium), en colaboración con Genasys, Loockheed Martin, Mitre, Oracle, Sedona Graphics y Sun Mycrosystems, que son varios de los proveedores de tecnología GIS más importantes. La aplicación se ha desarrollado para conseguir la interoperatividad de datos y sistemas de GIS y ha sido presentada a los miembros técnicos de la comunidad OGC en respuesta a una petición que éstos realizaron para aprobar la implementación de CORBA.

La presentación podría conseguir, si la OGC lo acepta, que CORBA fuera elegido como medio de comunicación para compartir la información GIS en un entorno de red, debido a varias razones, como por ejemplo que su arquitectura de soporte multiplataforma es compatible con sistemas GIS ya existentes, con servidores de datos y clientes de Internet.

La implementación de CORBA está basada en tecnología ProActiveM de Bentley y una versión orientada a objetos de MicroStation Geographics. Incluye especificaciones para la traducción de datos, datos compartidos en tiempo real y acceso remoto a operaciones geoespaciales tales como intersecciones y uniones. También ofrece un acceso completo a estas funcionalidades tanto a Java como a otras aplicaciones basadas en CORBA.

Para obtener más información sobre Bentley: http://www.bentley.com/ o llamar al teléfono 91-3728494.

Breves

LA EXPO 98 TENDRÁ UN CONTROL DE ACCESO ESPAÑOL

Será Informática El corte Inglés quien a través de su delegación permanente en Lisboa realizará la instalación del sistema de control de acceso y ticketing a la Expo 98.

Ya tuvo sus primeras experiencias en la Expo 92, las Olimpiadas de Barcelona y Porta Aventura y repetirá en Portugal en el evento deportivo.

La aplicación está probada y especializada para este tipo de eventos y además de un sistema de acceso es una herramienta muy útil y eficaz de gestión y emisión de billetes. Su integración se realizará en fases distintas: la venta anticipada de entradas en la oficina principal de venta, será la primera; le seguirá la instalación de las expendedoras automáticas repartidas por la ciudad y después el control de la venta de localidades dentro del recinto olímpico.

INFORMIX-ONLINE DYNAMIC SERVER

Informix Software ha puesto a disposición de sus clientes la versión 7.21 del servidor de bases de datos INFORMIX-Online Dynamic Server, optimizada para trabajar con los sistemas de Silicon Graphics, Inc. Esta optimización se ha conseguido en parte mediante la inclusión del soporte para direccionamiento de memoria VLM(Very Large Memory) de 64 bits.

Esta nueva versión proporcionará a los usuarios de sistemas de Silicon Graphics un aumento generalizado del rendimiento del sistema para aplicaciones que manejen grandes bases de datos (VLDB) y aplicaciones críticas de negocios, como OLTP y data warehousing.

EN EL MERCADO ILOG RULES PARA JAVA

Ilog presenta un sistema basado en reglas para los lenguajes C++ y Java

Ilog es una empresa que se dedica a desarrollar componentes de software avanzado, entre otras cosas, y ha creado un motor basado en reglas que se comercializa en todo el mundo. ILOG Rules permite la manipulación de objetos nativos de Java mediante una sintaxis que se basa en reglas; el compilador de ILOG Rules convierte estas reglas en una clase de agente inteligente cuyas instancias se integran dentro de la aplicación o applet. Estos agentes reaccionan seleccionando las acciones según los estados que presentan los objetos Java.

Uno de los factores que incrementa el rendimiento de ILOG Rules podría ser la implementación del algoritmo RETE, que optimiza los tiempos de respuesta y minimiza el requerimiento de memoria para los agentes.

El producto se encuentra ya disponible sobre Visual C++, Visual J++ y Visual Café sobre Windows 95 y Windows NT, así como para c++ Y Java sobre Solaris, HP-UX, AIX y otras plataformas. Se puede adquirir una licencia de desarrollo sobre PC desde un millón de pesetas.

UNIVERSAL TOOLS STRATEGY

Nueva estrategia de Informix para el desarrollo de herramientas para sistemas abiertos

Respaldada por Microsoft, Symantec, JavaSoft, PowerSoft, Forte y Seagate Software, Informix tiene como objetivo simplificar la construcción de aplicaciones para Universal Server, utilizando las herramientas que estas compañías lanzan al mercado.

La estrategia Universal Tools pretende conseguir que las aplicaciones generadas con las herramientas de las principales compañías del sector soporten de forma nativa y dentro de un entorno de sistemas abiertos los tipos de datos que gestiona el servidor Universal Server. Dentro de esta estrategia se incluye el lanzamiento de una herramienta nueva, Informix-Data Director, que incrementará la productividad en el desarrollo de aplicaciones y cubrirá distintos entornos de desarrollo del mercado como son: Web/Java, Windows y entornos de desarrollo cliente/servidor como NewEra, Forte o PowerBuilder.

Estos entornos junto con Data Director proporcionan funcionalidad drag-and-drop y model-driven para extender, explotar y gestionar distintos tipos de aplicaciones, ya sean Java, Web, Windows o distribuidas cliente/servidor, directamente en el servidor.

SYBASE IBERIA PRESENTÓ SUS NOVEDADES EN TECNOLOGÍAS Y PRODUCTOS

El día 21 de mayo tuvo lugar en Madrid la celebración del evento ImpactNow organizado por la empresa Sybase Iberia como fondo para la presentación de las novedades en sus productos. Como patrocinadores además contaron con la presencia de Hewlett-Packard y Sun Microsystems, y en el Hall de Exposición contó con empresas como ADD Servicios Informáticos, Aleph Software, Arcisa, Brio Technologies, Business Objects, Compaq, Cyrano, Eagle, Digital, Empresarios Agrupados, Eti, K&P, Peoplesoft, Servicios Informáticos, Pullervi, SELESTA, SMS, Summit y Seagate Software.

En ImpactNow pudimos asistir a tres sesiones en paralelo: NetImpact, WarehouseNow y SolucionesNow donde, mediante una serie de ponencias y exposiciones, se dieron a conocer las últimas tecnologías para Internet/Intranet y Data Warehousing de la compañía, los organizadores y las empresas expositoras.

Las diferentes ponencias cubrieron distintas áreas, como el desarrollo de aplicaciones transaccionales y el desarrollo RAD de aplicaciones en Internet o Java en el mundo empresarial, donde se presentó una variada oferta de productos Sybase orientados a facilitar el desarrollo en Java así como las posibilidades que tiene Java como entorno para crear aplicaciones reales de negocio. También pudimos escuchar ponencias sobre *Warehousing*, donde conocimos una nueva generación de servidores y productos de conectividad para ayudar a las empresas en el análisis de la información. Tecnologías como MBDMS, OLAP, ROLAP y tecnologías de indexado y almacenamiento en el servidor demostraron los beneficios que se pueden obtener con su utilización.

INFORMIX OFRECE EN NEWERA 3.1 SOPORTE PARA UNIVERSAL SERVER

También anuncia que NewEra 3.0 estará disponible para Windows 95 y Windows NT

Informix mejora NewEra para facilitar la construcción de aplicaciones capaces de tratar los tipos de datos que gestiona Universal Server.

El entorno de desarrollo gráfico orientado a objetos NewEra ofrecerá la capacidad de crear aplicaciones cliente/servidor escalables, que puedan acceder a la información almacenada en los gestores de bases de datos Informix basados en DSA (Dynamic Schalable Architecture), tanto Universal Server como la familia de servidores Online. Con las mejoras del entorno, se incluirá el soporte para Windows 95 y Windows NT, además de soporte nativo para desarrollar aplicaciones para Universal Server. Como añadido el entorno también ofrece un conjunto de herramientas para incrementar la productividad del desarrollador en entornos Microsoft de 32 bits y una interfaz visual mejorada.

NewEra 3.0 está ya disponible, mientras que la disponibilidad de la versión 3.1, con capacidades Web y soporte para Informix Universal Server, se prevé para el tercer trimestre de este año.

Para más información: Informix Software Ibérica, Natividad de Mateo, (91) 372 98 00.

El Pentium II forma el núcleo de la familia PC300XL de IBM

El Gigante IBM anuncia nuevos equipos de trabajo con microprocesadores Pentium II de 233 o 266 Mhz, para proporcionar soluciones para aplicaciones de software como Internet/Intranet, multimedia y comunicaciones. No han tardado demasiado en incorporar estos nuevos procesadores para conseguir ofrecer prestaciones que faciliten la gestión integrada de los usuarios corporativos, un alto nivel de seguridad y una protección de datos superior.

Los nuevos sistemas de IBM incorporan capacidades de soporte para gráficos de 64 bits (con tarjeta estándar Trio 64 V2 de S3 con 2 Mb de VRAM) y soporte de vídeo a pantalla completa integrado en la placa base. Windows 95 y Windows NT son los sistemas operativos elegidos como precargados y como opciones se tienen DOS, Windows 3.11 y OS/2 Warp 3.0 ó 4.0. Además es posible elegir discos duros BusMaster EIDE (entre 2,5 GB y 4,2 GB) o bien unidades Ultra SCSI de 4,3 GB.

La familia PC 300XL está disponible en España en siete modelos con precios desde 478.000 pesetas (son monitor).

Para más información sobre la compañía, sus productos y servicios consultar: www//www.ibm.com. Para conseguir la información específica sobre PCs IBM consultar la página http://www.pc.ibm.com. Como teléfono de referencia para usuarios: 900 100 400.

Breves

"Búscalo en Castellano... en Internet"

Mediante el programa de colaboración "Búscalo en Castellano... en Internet", Microsoft firma acuerdos con los principales buscadores españoles en Internet, para conseguir que los usuarios de Internet puedan buscar en español.

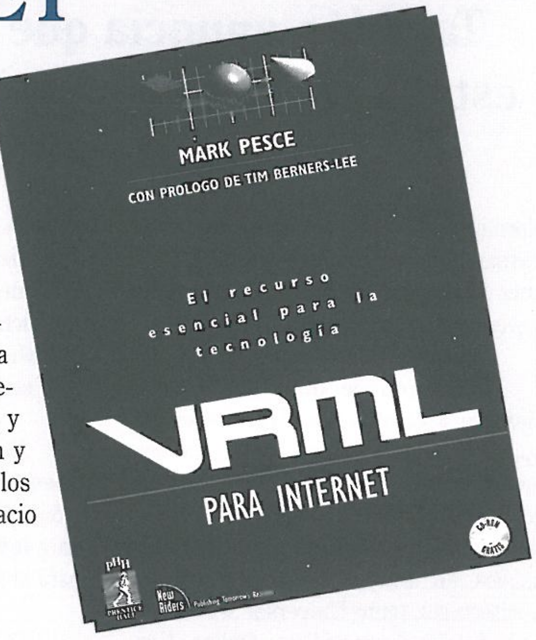
Microsoft ha facilitado la información, la formación y la tecnología para que los buscadores puedan implantar servicios en torno a sus dos estrellas Microsoft NetMeeting y Microsoft Chat, dentro de Microsoft Internet Explorer.

La nueva página de búsqueda de la compañía, disponible en la dirección http://www.microsoft.com/spain/ie/amigos/buscar.htm permite a los internautas la búsqueda de la información de forma fácil y directa en los buscadores que han firmado el acuerdo como son Biwe, OZU, OLE, Trovator, El Cano, El inspector de Telépolis, o Vindio.

NUEVO MÓDULO DATABLADE PARA TRATAMIENTO DE IMÁGENES

En su política de desarrollo de módulos software para Universal Server de Informix, la compañía, en colaboración con Eastman Kodak, ha anunciado la disponibilidad del nuevo módulo FlahsPix, que permitirá almacenar, administrar y utilizar imágenes en alta definición dentro de bases de datos relacionales. El módulo proporcionará a los usuarios varias funcionalidades FlahsPix, como la posibilidad de leer o escribir en múltiples resoluciones una misma imagen, o la extracción e indexación automática de la información y la encriptación de datos descriptivos con archivos de imagen.

VRML PARA INTERNET



Términos como espacio cibernético llevan de moda algún tiempo y se utilizan tanto que casi no tienen un significado definido. VRML es un lenguaje para modelado de realidad virtual, lo cual es tanto como decir que es un lenguaje para intentar comunicar a los demás la imaginación del programador o del diseñador que inventa el espacio cibernético que crear.

A través de tres trayectorias propuestas por el autor, Mark Pesce, en este libro se nos muestran los conceptos básicos del VRML; estas trayectorias incluyen una para los principiantes, donde se trata el contexto básico, cómo navegar en el espacio cibernético y la preparación de gráficos; la trayectoria para el desarrollador es adecuada para aquellas personas que tienen experiencia en el manejo de la Web y en la utilización de gráficos 3D, y trata temas como VRML avanzado, optimización, aspectos de publicación y cómo elaborar un visualizador VRML; la trayectoria del diseñador abarca los gráficos 3D, herramientas y aspectos de publicación y el diseño en el espacio cibernético.

Editorial: Prentice Hall
Autor: Mark Pesce
421 páginas
Idioma: español
Incluye CD-ROM

LA ESENCIA DE VISUAL BASIC 4



Enfocado hacia los programadores novatos y aquellos más experimentados que deseen desarrollar aplicaciones para Windows utilizando Visual Basic 4.0, este libro pretende proporcionar un tratamiento práctico de la programación para Windows, con muchos ejemplos de cómo hacerlo y sin exceso de teoría.

A través de temas como el lenguaje y su entorno; exploración de los controles de la caja de herramientas; creación de una aplicación, desde el proyecto hasta el ejecutable; trazos, animación y juegos; programas de bases de datos; manejo y depuración de errores; temas avanzados de programación y la creación de archivos de ayuda para Windows 95, la obra es un interesante material de referencia para afianzar los conocimientos de todo programador de Visual Basic.

Editorial: Prentice Hall
Autor: Mark Steven Heyman
457 páginas
Idioma: español
Nivel: avanzado

JavaScript: Frames y Windows

WWW

Alejandro M. Reyero Abad
axl@las.es

Las páginas con frames y múltiples ventanas son la última moda en el mundillo WEB, son elegantes, prácticas y se prestan a una variedad de efectos hasta ahora desconocida. En este artículo veremos cómo crearlas y darles un poco de vida. Con JavaScript, por supuesto.

Antes de explicar cómo funcionan juntos los frames y JavaScript vamos a explicar qué son los frames y para qué se utilizan.

Las ventanas del Browser (tanto la principal como las creadas con el objeto *window)* pueden ser divididas en varias secciones o frames, siendo cada uno de ellos una zona independiente (aunque relacionadas entre sí, como veremos más adelante) dentro de la ventana.

Para crear frames vamos a necesitar dos *TAGS* HTML: <FRAMESET> y <FRAME>.

El primero define las características generales de la página (tamaño de los frames, si están divididos por columnas o filas, etc.) y el segundo hace referencia a cada frame en particular. La página dos.htm es un ejemplo bastante simple de utilización de frames:

Este ejemplo divide la pantalla en dos partes iguales, presentando en la parte superior la página WEB de Netscape y en la inferior la de Microsoft, siendo el ejemplo más simple de frames.

Con *<FRAMESET rows="50%, 50%">* dividimos la ventana en dos filas iguales (50% de la ventana cada una). Si quisiéramos dividirla en columnas sólo habría que sustituir el *rows* por *cols*. El tamaño de los frames puede ser indicado tanto en porcentaje (como acabamos de ver), como en píxeles (indicando el número de píxeles y omitiendo el símbolo %), además podemos usar un asterisco para que la página "coja" todo el espacio restante, por ejemplo, con *<FRAMESET COLS="150,*,200">* crearíamos tres frames en nuestra ventana, repartidos en 3 columnas, siendo la primera de 150 píxeles, la tercera de 200 y la del medio cogerá el espacio restante entre ambas.

El *TAG <FRAME ...>* indica la página a cargar en ese frame, el cual puede tener un nombre (fundamental a la hora de hacer referencia a ellos con JavaScript) que indicaremos con la propiedad *name*. El *TAG <FRAME>* tiene otras propiedades, como *noresize,* con el que indicaremos al *browser* que no permita al usuario

Dos.htm

```
<HTML>

<FRAMESET rows="50%,50%">
<FRAME
    src="http://home.netscape.com">
<FRAME
    src="http://www.microsoft.com">
</FRAMESET>


</HTML>
```

cambiar el tamaño del frame, *scrolling=yes/no/auto* que especifica si el frame va a tener barras de desplazamiento (si, no o automático) y *marginheight=valor* y *marginwidth=valor* que dejan al usuario especificar, en píxeles, los márgenes superior, inferior, derecho e izquierdo del frame. Si los márgenes especificados son imposibles de presentar en el frame, son ignorados.

También se pueden tener varios *FRAMESETS* anidados para incluir más frames dentro de otro frame.

La página *cuatro.htm* es un ejemplo:

Cuadro.htm

```
<HTML>

<FRAMESET cols="50%,50%" border=0>

<FRAMESET rows="300,*">
<FRAME
    src="http://home.netscape.com"
    name="Netscape">
<FRAME
    src="http://www.microsoft.com"
    name="MicroSoft">
</FRAMESET>

<FRAMESET rows="60%,40%">
<FRAME
    src="http://www.3drealms.com"
    name="3DRealms">
<FRAME
    src="http://www.idsoftware.com"
    name="IDSoftware">
</FRAMESET>

</FRAMESET>

</HTML>
```

En *cuatro.htm* se divide la ventana en cuatro frames, usando dos *FRAMESET* dentro de otro *FRAMESET*. Nótese el uso de *border=0* en el primer *FRAMESET*, que indica que todos los frames incluidos dentro del mismo no van a tener borde, el cero se puede sustituir por el tamaño de borde que deseemos (solo a partir de Netscape 3.0 y superiores).

Frames y JavaScript

Veamos ahora cómo JavaScript se entiende con los frames en una ventana. Usaremos un ejemplo simple, un *FRAMESET* con dos *FRAMEs*, el primero llamado *frame1* que incluirá una página llamada *primera.htm* y el segundo *frame2*, que contendrá la página *segunda.htm*.

```
<frameset cols="50%,50%">
    <frame src="primera.htm" name="frame1">
    <frame src="segunda.htm" name="frame2">
</frameset>
```

Como ya hemos visto en anteriores artículos, JavaScript organiza jerárquicamente todos los elementos de una página WEB, también los frames. En la figura 1 veremos la jerarquía de este ejemplo.

Vemos que los dos frames tienen el mismo padre: la ventana principal.

Podemos tener varios FRAMESETs anidados para incluir más frames dentro de otro frame

Como nuestros frames tienen nombre (*frame1 y frame2*) podemos intercambiar información entre ambos. Se pueden dar tres casos:

- La ventana padre accede a un frame hijo.
- Un frame hijo accede a la ventana padre.
- Un frame hijo accede a otro frame hijo.

Como vemos en la figura 1, la ventana padre está directamente conectada con todos sus frames, si desde ella quisiésemos acceder a uno de los frames, únicamente tendríamos que usar el nombre del frame. Por ejemplo, para escribir una frase en *frame1*:

```
frame1.document.write("Estamos escribiendo
    desde la ventana padre");
```

A veces tendremos que acceder a la ventana padre desde uno de los frames hijo, por ejemplo, para quitar los frames de

Figura 1. Jerarquía simple.

Ventana Principal
PADRE
frame1 primero.htm
frame2 segundo.htm
HIJOS

la página, cargando una nueva que sustituya a la anterior.

Para acceder a la ventana padre usaremos *parent* desde la ventana hija. Vemos *primero.htm* como ejemplo.

Primero.htm

```
<HEAD>
<SCRIPT>
function nosVamos(){
    parent.location.href="http://
        www.las.es/ grasa";
}
</SCRIPT>
</HEAD>

<FORM>
    <INPUT TYPE="button"
        VALUE="Pinchame"
        onClick="nosVamos()">
</FORM>
```

Para cargar un nuevo documento se asigna un URL a *location.href,* como queremos quitar los frames de la página usamos el objeto *location* de la ventana padre, en nuestro ejemplo vamos al fancine electrónico GRASA:

```
parent.location.href="http://www.las.es/ grasa";
```

Finalmente, también podemos acceder desde un frame hijo a otro frame hijo. Observando con detenimiento el esquema anterior notaremos que no existe conexión directa entre ambos, no se puede llamar a *frame2* directamente desde *frame1* (y viceversa) ya que no conocen sus respectivas existencias. ¿Tiene esto solución? Sí, muy sencilla.

Hemos visto que desde el frame hijo podemos acceder a la ventana padre y desde ésta, lógicamente, se pueden acceder a todos los frames que contiene. Para escribir en *frame1* desde *frame2* usaríamos:

```
parent.frame1.document.write("Estamos
    escribiendo entre ventanas.");
```

Frames y barras de navegación

Como ejemplo práctico de utilización de frames, vamos a dotar a nuestra página WEB de una "Barra de navegación" (Navigation bar), es decir, tendremos un número X de frames en nuestra página, pero uno de ellos siempre va a permanecer inalterable, conteniendo links que actualizarán al resto de los frames.

Lo primero que necesitaremos es la página índice, en la que incluiremos los *FRAMESETs* necesarios para presentar todos los frames.

Index.htm

```
<HTML>

<FRAMESET ROWS="80%,20%">
<FRAMESET COLS="125,*">
    <FRAME src="izq.htm"
    name="izquierda">
    <FRAME src="der.htm"
    name="derecha">
</FRAMESET>

<FRAME src="menu.htm"
name="menu">

<NOFRAME>
Lo siento, esta p&aacute;gina contiene
frames y tu browser no los
soporta.
</NOFRAME>
</FRAMESET>
</HTML>
```

Vemos que nuestra página índice (*index.htm*) va a contener 3 frames: *izquierda, derecha* y *menu,* siendo este último el que va a contener nuestra barra de navegación, *izquierda* presentará animaciones y referencias dependiendo de la información a visualizar, y *derecha* será el frame "principal", en el que aparecerá el grueso de la información.

Antes de proseguir, conviene reseñar la presencia del TAG *NOFRAME*, cuyo contenido será ignorado por los *browsers* capaces de visualizar frames, e impreso en la ventana principal en los que no lo son.

Los contenidos de *derecha* e *izquierda,* tras ser cargados por *index.htm,* son triviales (esto es un ejemplo, no vamos a desarrollar un proyecto profesional) como vemos en sus respectivos códigos fuente.

Izq.htm

```
<HTML>

<CENTER>
<P><FONT size=+2
    color="red">izq.htm</FONT></P>
</CENTER>

</HTML>
```

Der.htm

```
<HTML>
<P><FONT size=+2
    color="blue">der.htm</FONT></P>
<CENTER>
<IMG src="soloprog.gif" width=500
    height=325>
</CENTER>

</HTML>
```

La parte más importante del ejemplo es *menu.htm*, nuestra barra de navegación.

Menu.htm

```
<HTML>
<HEAD>
<SCRIPT>
function soloCarga(){
parent.izquierda.location.href="soloizq.
    htm";
parent.derecha.location.href="soloder.
    htm";
}

function programadoresCarga(){
  parent.location.href="programa.htm";
}
function ventanasCarga(){
  parent.derecha.location.href="venta-
nas.htm";
}
</SCRIPT>
</HEAD>
<FORM>
```

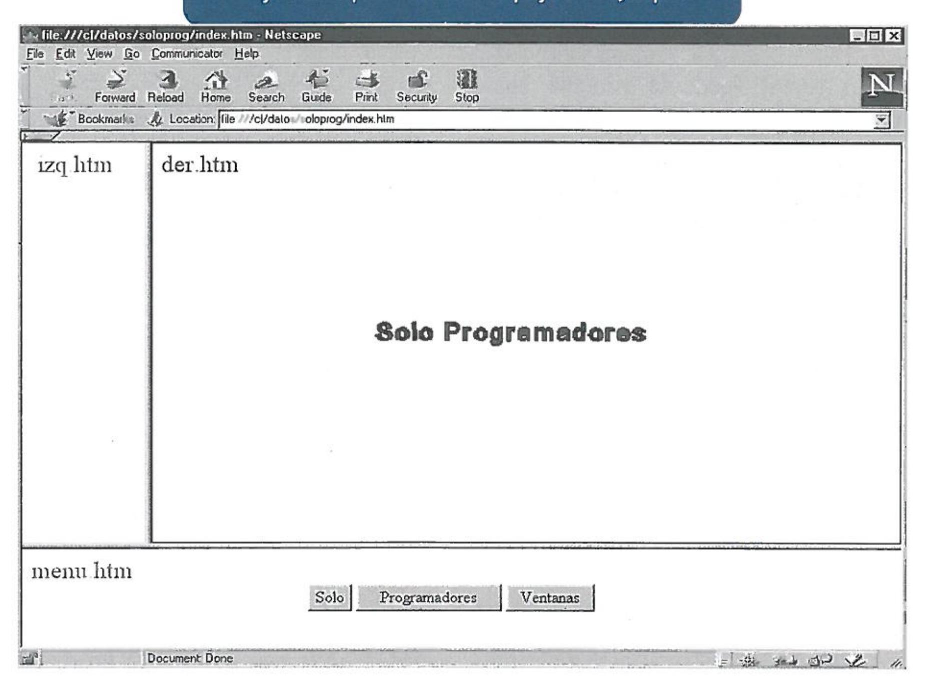
Consiste en un form con tres botones, cada uno de ellos llama a una función JavaScript (utilizando el método *onClick*) que actualizará uno o varios frames:

SoloCarga(): llamado por el botón *Solo*. Actualiza los frames *izquierda* y *derecha*, cargando dos páginas bastante simples que no vamos a comentar (este ejemplo está completo en el CD).

ProgramadoresCarga(): llamado por el botón *Programadores*. Utiliza una llamada directa a *parent* para actualizar la página principal, descargando todos los frames (*parent.location.href= "programa.htm";*).

VentanasCarga(): únicamente actualiza el frame *derecha*, cargando la página *ventanas.htm* que comentaremos más adelante.

Figura 2. Aspecto de nuestra página de ejemplo.



Con JavaScript podemos actualizar más de un frame a la vez

Como acabamos de ver, actualizar frames (incluso más de uno simultáneamente) mediante JavaScript es tarea fácil, aunque conviene recordar la posibilidad de hacerlo mediante HTML (sólo un frame cada vez) añadiendo TARGET a la llamada a la página, por ejemplo:

```
<A HREF=http://www.javasoft.com
    TARGET="derecha">
```

cargaría la página de JavaSoft (división de SUN encargada del desarrollo del lenguaje Java) en el frame llamado *derecha*. Para que la página eliminara los frames y se cargara en la ventana principal tendríamos que sustituir el *TARGET="derecha"* por *TARGET= "_TOP"*.

Abriendo ventanas

La última moda en páginas WEB consiste en la utilización de varias ventanas para presentar una única página. De la conjunción de ventanas JavaScript, frames y unos buenos diseñadores gráficos, pueden surgir auténticas maravillas como la WEB de la cadena musical MTV (www.mtv.com).

Para abrir nuevas ventanas usaremos el objeto *window*, que es el objeto principal para cada grupo de objetos *location, document* e *history.*

Para definir una ventana (*window*), usaremos el método *open*:

```
ventana = window.open("URL",
"nombre_de_la_ventana"
[,"Parámetros, de, la, ventana"]);
```

En el que:

ventana es el nombre de la nueva ventana. Usamos esta variable cuando nos referimos a las propiedades, métodos y contenidos de la ventana.

URL es la página a cargar en la ventana.

nombre_de_la_ventana es el nombre con el que vamos a hacer referencia a la ventana con el atributo *TARGET* en los *TAGs* *<FORM>* y *<A>*.

Parámetros de la ventana es una lista (separada por comas) de alguna de las siguientes opciones:

```
toolbar[=yes|no]|[=1|0]
location[=yes|no]|[=1|0]
directories[=yes|no]|[=1|0]
status[=yes|no]|[=1|0]
menubar[=yes|no]|[=1|0]
scrollbars[=yes|no]|[=1|0]
resizable[=yes|no]|[=1|0]
width=pixels
height=pixels
```

Los parámetros "Booleanos" (los que requieren yes/no o 1/0) toman el valor *true* si se especifican sin valores, o como *yes* o 1.

Se pueden usar uno, varios o todos los parámetros simultáneamente, teniendo en cuenta que irían separados por comas y sin espacios entre ellos. Por ejemplo:

```
MiVentana = window.open("esta.htm",
    "estaVentana",
    "scrollbars=
    yes,width= 300,height=400");
```

Vemos con detenimiento la utilidad de todos los parámetros:

toolbar crea, si su valor es *true,* los botones estándar del *browser* (*BACK, FORWARD, STOP, RELOAD, etc.*).

ocation crea el campo de introducción del URL, esto es, el *Text Box* del *browser* en el que ponemos la dirección de la página a visitar.

directories crea los botones de navegación del *browser*, en Netscape 4.0 son *What's New* y *What's Cool.*

status crea la barra de estado que esta en la parte inferior del *browser*.

menubar crea el menú que está en la parte superior del *browser.*

scrollbars crea las barras de desplazamiento horizontales y verticales cuando el documento que visualizar es más grande que la ventana. Si esta opción es *false* y la ventana no puede contener todo el documento, no podremos visualizarlo al completo.

resizable permite (si es *true*) al usuario poder cambiar el tamaño de la ventana con el ratón.

width especifica el ancho de la ventana en píxeles.

height especifica el alto de la ventana en píxeles.

Por tanto, nuestro anterior ejemplo, *MiVentana*, abriría una ventana de 300 píxeles de ancho por 400 de alto, con barras de desplazamiento.

Ventanas.htm

```
<HTML>
<HEAD>
<SCRIPT>
function ventana1(){
    primVentana=window.open("http://www.allgames.com/XvT",
    "allgames", "scrollbars=yes,width=625,height=450");
}
function ventana2(){
    segVentana=window.open("","vacio",
    "scrollbars=no,resizable=no,width=400,height=400");
    segVentana.document.writeln("<H2>Escribo desde otra ventana</H2>");
}
function ventana3(){
    tercVentana=window.open("cerrar.htm","cerrar","width=500,height=400");
}
</SCRIPT>
</HEAD>
<P><FONT size=+2 color="blue">ventanas.htm</FONT></P>
<MENU>
<LI><A HREF="javascript:ventana1()">Ventana 1</A>
<LI><A HREF="javascript:ventana2()">Ventana 2</A>
<LI><A HREF="javascript:ventana3()">Ventana 3</A>
</MENU>
</HTML>
```

Una vez conocido el objeto window, y todas sus propiedades, vamos a comentar el ejemplo de creación de ventanas de nuestra página, *ventanas.htm*

En esta ocasión no utilizamos un *onClick* para llamar a las funciones, ya que el método de selección es un *<MENU>* con varios *TAGs* referencia *(<A ... >)*.

Para llamar a una función desde una referencia, usaremos el "protocolo" *javascript:*, es decir:

```
<A HREF="javascript:funcion_a_llamar()" >
    Texto </A>
```

La primera función, *ventana1*, abrirá una ventana llamada "allgames", en la que cargará la página de *All games network* dedicada al último juego de la saga *Star Wars, X-Wing vs. TIE-Fighter*. Esta ventana tendrá barras de desplazamiento y su tamaño será de 625 por 450 píxeles.

La función llamada *ventana2* estará, en principio, vacía. No cargaremos en ella ninguna página, ni local ni remota. En cambio escribiremos en ella desde *ventanas.htm* (en la misma función *ventana2*) con

```
segVentana.document.writeln("<H2>Escribo
    desde otra ventana</H2>");
```

La última ventana es aún más simple, cargando en ella la página local llamada *cerrar.htm*, siendo el tamaño de la ventana las únicas propiedades que utilizaremos.

Vemos en el código fuente de *cerrar.htm* una de las maneras de cerrar una ventana.

Cada una de las siguientes instrucciones cerrará la ventana actual:

```
window.close()
self.close()
close()
```

Para cerrar una ventana llamada *otraVentana* usaríamos:

```
otraVentana.close()
```

Conclusiones

Después del artículo aparecido en el número 33 de Sólo Programadores y de éste que acabamos de terminar, ya estamos preparados para utilizar JavaScript a un nivel bastante interesante, ya que conocemos la jerarquía de objetos de JavaScript, sabemos utilizar el objeto *window* y la relación entre *frames* y

Actualizar frames mediante JavaScript es tarea fácil, aunque conviene recordar la posibilidad de hacerlo mediante HTML añadiendo TARGET a la llamada a la página

JavaScript no es ningún secreto para nosotros.

A partir de ahora profundizaremos en las opciones avanzadas de JavaScript, conociendo nuevos objetos predefinidos, utilizando LiveConnect para que nuestros *APPLETS* Java se entiendan con nuestro código JavaScript, aprendiendo a utilizar opciones disponibles en la última actualización del Netscape Navigator (4.0) como son los Layers, etc...

Contactar con el autor

Si el lector quiere hacer llegar sus opiniones, preguntas, ejemplos, o, simplemente, decir "hola" al autor:

E-Mail: axl@las.es
WEB: http://www1.las.es/~axl

Cerrarhtm

```
<HTML>
<FORM>
<INPUT TYPE="button" value="Cierra" onClick="window.close()">
</FORM>
</HTML>
```

Bibliografía y referencias

- "JavaScript: Una chispa de vida WEB", por Fernando J. Echevarrieta. Solo Programadores número 30.
- "El Modelo de Objetos de JavaScript", por Alejandro M. Reyero. Solo Programadores número 32.
- "The JavaScript Forum", http://www.geocities.com/ResearchTriangle/1828/index.html .
- "No Content", http://www.btinternet.com/~martin.webb.

Widgets con Tk

Jose Antonio Collar Ruano

El Toolkit de Tk incorpora los servicios gráficos básicos para construir las más potentes aplicaciones, al modo de los innovadores entornos de programación gráfica. Estos "ladrillos" lógicos son lo suficientemente sencillos y manejables como para poder fundar construcciones más complejas combinando algunos de ellos. Vamos a recorrer a continuación varios de estos elementos, aquellos que en primer lugar necesitaremos para desarrollar una aplicación simple.

Etiquetas de texto

El siguiente programa crea una etiqueta de texto.

```
label .etiqueta1 -text "Aquí tenemos nuestra
        primera etiqueta de texto"
pack .etiqueta1
```

El comando *label* produce la generación de la etiqueta, mientras que el comando *pack* se encarga de mostrar la etiqueta creada.

Las etiquetas se usan para enlazar texto a otros widgets, como veremos más adelante.

Este programa tan simple puede ser modificado para mostrar otras capacidades, incluyendo colores de fondo y de primer plano, así como cuadros alrededor del texto.

Por ejemplo:

```
label .eticuadro -text "Otro texto para continuar"
        -relief sunken
pack .eticuadro
```

proporciona una apariencia de cuadro a la etiqueta de texto, del mismo modo que:

```
label .eticolor -text "Añadamos un poco de color"
        -background red -foreground blue
pack .eticolor
```

genera texto con los colores especificados, rojo para el fondo y azul para el primer plano.

El siguiente ejemplo muestra cómo se puede usar una variable Tcl para el texto:

```
label .etivar -text $texto
pack .etivar
```

Para ejecutar este ejemplo lanzaremos en primer lugar el comando *wish* y después iniciaremos el widget con el comando *source*.

```
wish
        set texto "Lo que bien empieza"
        source etivar.tcl
        set texto "bien termina".
```

En el ejemplo anterior, el texto de la etiqueta es constante y no cambia cuando se modifica la variable *texto*.

Los widgets son los componentes preconstruidos que componen el armazón del Toolkit de Tk. Su sencillez de programación junto con la potencia y la flexibilidad de sus propiedades los hacen imprescindibles en cualquier aplicación gráfica Tcl.

Figura 1. Configuación de los widgets de etiquetas.

```
#Este ejemplo muestra diferentes formas para los widgets de etiqueta
    label .lab1 -text "De momento sólo texto"
    label .lab2 -text "Con un poco de color" -foreground red
            -background blue
    label .lab3 -text "Ahora deprimido"  -relief sunken
    label .lab4 -text "De otro modo, compruébalo" -relief groove
    label .lab5 -text "Texto plano"    -relief flat
    label .lab6 -text "En cadena"  -relief ridge
    label .lab7 -text "Elevado"  -relief raised
    label .lab8 -text "Times, una fuente elegante"
            -font *-times-medium-r-normal--*-100-*
    pack .lab1 .lab2 .lab3 .lab4 .lab5 .lab6 .lab7 .lab8
            -padx 2m -pady 1m -fill x
```

Cada tipo de widget lleva asociado un gran número de opciones deconfiguración, en relación a su apariencia, al modo de manejar los datos, la forma de comportarse frente a determinados eventos, etc.

Si la etiqueta se especifica con el uso de la opción *textvariable*, como en el siguiente ejemplo:

```
label .etivar -width 20 -textvariable texto
```

los cambios en la variable *texto* se verán reflejados en la etiqueta. Hay que resaltar que el cambio en la etiqueta se produce cuando se ejcuta el segundo comando *set*.

El último ejemplo (ver figura 1) de esta sección muestra la mayoría de las posibilidades de configuración de los widgets de etiquetas. Con él introducimos lo que es común para todos ellos, es decir, el gran número de opciones que nos permiten mantener un control riguroso de la apariencia y el comportamiento de nuestras aplicaciones.

Mensajes

Los widgets de mensaje son similares a widgets de etiqueta, puesto que muestran texto, aunque éste puede ocupar varias líneas.

```
message .msg -text "Los antiguos griegos se
        sentaban cómodamente
sobre los asientos del auditorium,
mientras que la orchestra los coros se deslizaban
        como un sólo hombre.
El proskenion era el reino de los actores, que
        usaban
la skene para el cambio de máscaras.
Los grandes festivales se prolongaban
durante días, y en ellos participaba
gran cantidad de público, que decidía en último
        término
la suerte de los autores.

La enormidad de las construcciones teatrales
condujo a un problema difícil de resolver:
la acústica.
La calidad de la colina sobre la que se asentaría
el nuevo teatro era fundamental para los
        arquitectos."
pack .msg
```

Está permitido insertar caracteres de continuación, tabuladores (t) y caracteres de fin de línea (n), dentro de un widget de mensaje para un mejor control sobre la salida, como en el siguiente ejemplo:

```
message .msg -width 400  -text "Los teatros grie-
        gos se formaban con los siguientes elemen-
        tos   principales: \n
t * El auditorium \t\t para el cómodo asiento de los
        espectadores.
\t * La orchestra \t\t dónde se desenvolvía el coro
\t * El proskenion \t\t zona reservada a los actores
\t * La skene \t en principio sólo utilizada
para el cambio de máscaras \n
La enormidad de las construcciones teatrales
condujo a un problema difícil de resolver:
la acústica.
La calidad de la colina sobre la que se asentaría
el nuevo teatro era fundamental para los
        arquitectos."

pack .msg
```

Los widgets de etiquetas de texto y los de mensaje poseen un conjunto de valores configurables bastante similar, como -relief sunken, etc.

Es muy útil mantener un control explícito sobre el ancho y la altura de las ventanas de mensaje, ya que ayudará a controlar la apariencia del texto.

Un ejemplo de este parámetro propio de este widget sería:

```
message .msg2
    -width 200 text
    "Mantengamos nuestra aplicación bajo
    control:
Que no se nos escapen de la ventana."
pack .msg2
```

Entrada de texto

Los widgets de entrada proporcionan cuadros de entrada para la introducción de datos y se definen como en el código siguiente:

```
entry .entr1 -textvariable variable
        pack .entr1
```

Aunque este cuadro de entrada es válido, es difícil de usar dado que carece de bordes marcados. El siguiente es más sencillo para el usuario:

```
entry .entr2 -relief ridge -textvariable variable
        pack .entr2
```

La opción de configuración *textvariable* asocia el cuadro de entrada con una variable Tcl.

Puede ser más adecuado determinar un ancho para el cuadro de entrada, dado que el valor por defecto puede resultar demasiado pequeño.

Esto se puede conseguir del siguiente modo:

```
entry .entr3 -width 30 -relief groove
        -textvariable variable
        pack .entr3
```

Cuadros de lista

Los widgets de cuadro de lista son similares en apariencia a los widgets de entrada, pero se diferencian en que los primeros permiten visualizar o seleccionar varias líneas de texto.

```
#Crea un cuadro de lista con los días de la semana
        listbox .lista1
        pack .lista1

#Lo que sigue es un comando específico del
        #widget cuadro de lista
#tinsert construye una lista de
#objetos comenzando en la línea 0
        .lista1 insert 0 lunes martes miércoles
        jueves viernes sábado domingo
```

Edición de texto

Los widgets de texto son elementos complejos que permiten introducir y editar múltiples líneas de texto. La edición se puede realizar situando el cursor dentro del texto con el ratón y tecleando o borrando caracteres según se desee. Permiten operaciones complejas, incluyendo la posibilidad de marcar texto dentro del cuadro para resaltarlo, así como el manejo de los eventos asociados al uso del ratón.

```
text .texto1 -width 45 -height 5 -relief sunken
        pack .texto1

#El siguiente es otro ejemplo de comando
        #específico de widget, en este caso para el
        #de edición de texto. insert crea una lista
        #de caracteres.
#Comienza en la línea 1 y el caracter 0

.texto1 insert 1.0 "    Sobre la maleza
        las brujas de Macbeth
        danzan en corro y gritan:
        tú serás rey."
```

Los widgets de texto proporcionan una herramienta poderosa para generar componentes de aplicaciones, como en el caso de los editores de texto.

Escala

Los widgets de escala permiten realizar la entrada directa de datos numéricos. El siguiente ejemplo crea un widget de escala y una etiqueta. El valor dispuesto por el widget de escala se muestra en la etiqueta. La escala se configura para mostrar un valor inicial.

```
#Crea una escala, orienteda verticalmente,
        #disminuyendo de 50 a 0.
#Se sitúan marcas cada 10 unidades.
        scale .escala1 -width 10 -orient vertical -
        length 280
            from 50 -to 0 -command "set var1"
            tickinterval 10 -relief raised

        label .lab1 -width 10 -textvariable var1 -relief
        sunken
```

Los enlaces permiten responder con acciones personalizadas a una combinación concreta de widget y evento del sistema, con lo que fundan un flujo de aplicación desde el desarrollo del interfaz gráfico

```
        pack .escala1 .lab1 -padx 1m -pady 2m -fill
        x

#Para establecer el valor de la escala con un co-
        mando del widget
        .escala1 set 25
```

Éstos son los widgets que podemos considerar esenciales y muchos otros integrados en diversas distribuciones se pueden considerar como variaciones sobre ellos.

Enlaces con widgets

Algunos widgets como los de entrada y los de texto mantienen un enlace implícito con su variable de texto, de tal modo que si los datos cambian dentro de la ventana, la variable de texto es actualizada automáticamente. Es así mismo posible enlazar explícitamente un widget con un evento para producir una acción.

Esto significa que siempre que el cursor se sitúe sobre un widget activo

Siempre que el cursor se sitúe sobre un widget activo y un determinado evento tenga lugar, se puede producir una acción adecuada

y un determinado evento tenga lugar, se puede producir una acción adecuada. Los eventos pueden ser de diversa naturaleza, tales como los derivados del movimiento del ratón, el uso de los botones del mismo, los clicks múltiples y la entrada desde el teclado. Como es obvio, se pueden mantener tantas acciones diferentes como eventos asociados al widget tengamos.

La instrucción para enlazar un widget a una acción es el comando *bind*. He aquí un ejemplo usando un widget de marco, que se utiliza principalmente como contenedor de otros widgets (ver figura 2).

El siguiente ejemplo (ver figura 3) es más complejo y muestra cómo se pueden realizar selecciones en un cuadro de lista utilizando el ratón.

Figura 2. Ejemplo de enlace de un widget a una acción.

```
frame marco1 -width 100 -height 100
    bind .marco1 "Button-1" {puts "marco1: Botón 1 presionado"}
    bind .marco1 "Button-2" {puts "marco1: Botón 2 presionado"}
    bind .marco1 "Button-3" {puts "marco1: Botón 3 presionado"}
    bind .marco1 "ButtonRelease-1" {puts "marco1: Botón 1 soltado"}
    bind .marco1 "Double-Button-1"
            {puts "marco1: Doble click en Botón 1"}
    bind .marco1 "Any-Motion" {puts "marco1: puntero en %x,%y"}
    pack .marco1
```

Figura 3. Ejemplo de selección en un cuadro de lista utilizando el ratón.

```
Crea un cuadro de lista con los días de la semana
#e implementa la posibilidad de seleccionar
    listbox .lista2 -exportselection 1
    pack .lista2

#Lo que sigue es un comando específico del widget cuadro de lista
#tinsert construye una lista de
#objetos comenzando en la línea 0
    .lista1 insert 0 lunes martes miércoles jueves viernes sábado domingo

#No hay enlaces predefinidos para el cuadro de lista
#Ahora enlazamos una acción al evento de soltar el Botón1 sobre un elemento de la lista
#Esto no interfiere con el proceso de selección
    bind .lista2 "ButtonRelease-1" {showindices}

#Ahora definimos el procedmiento para mostrar los índices
    proc showindices {} {
        puts [.lista2 curselection]
    }
```

Se debe ser cuidadoso con las acciones asociadas por defecto a determinados eventos con algunos widgets, ya que si asociamos una acción nueva a estos eventos, sobreescribimos y anulamos el anterior.

La herencia en el modelo orientado a objetos, del análisis a la implementación

José J. Merseguer y José Romero Cuñat

El artículo tratará la herencia desde dos puntos de vista, su formulación conceptual y su uso desde una perspectiva metodológica. Los conceptos serán plasmados con ejemplos en los lenguajes de programación VisualC++ y Delphi.

De entre el amplio abanico de metodologías OO que se ofrecen hoy en día en el mercado (OMT, Booch, Yourdon,...) hemos escogido la de Rumbaugh -OMT- como referencia por varias razones. Entre ellas, ser de dominio público (no propietaria), ser una de las más conocidas y utilizadas, así como disponer de uno de los CASE (Rational Rose) más exitosos y vendidos. OMT (Técnica de Modelado de Objetos) se define según su autor como "*una metodología OO para el desarrollo de software que se extiende desde el análisis hasta la implementación pasando por el diseño*".

La gran mayoría de las metodologías de análisis y diseño OO señalan la reutilización de código como una de las características más importantes entre todas las que ofrecen. En el modelo OO, la herencia es el mecanismo de abstracción básico para tratar, tanto desde el enfoque metodológico como en programación, la reutilización de código.

La herencia se puede plantear como un tipo de *polimorfismo*. Éste se puede precisar como la capacidad de definir entidades que pueden tomar más de una forma. Existen distintas variantes de polimorfismo. Por una parte, se encuentra el universal (considerado verdadero) en el que el código a ejecutar es constante. Por otra parte, está el polimorfismo ad-hoc (considerado aparente) donde se ejecuta código distinto para tipos de argumentos distintos.

Esquemáticamente:

- Polimorfismo universal.
  - Paramétrico: caso de las plantillas de C++ (llamado también Genericidad).
  - De inclusión: caso de la **Herencia.**
- Polimorfismo ad-hoc.
  - Sobrecarga de funciones.
  - Coerciones: caso de conversiones implícitas de tipos.

El artículo se estructura en tres apartados. En el primero de ellos veremos la herencia simple, continuaremos el siguiente con la herencia múltiple, el tercer apartado dará un repaso a las clases abstractas y para terminar se mostrarán las conclusiones del trabajo.

Herencia simple

Antes de dar una definición del concepto de herencia, pongamos un ejemplo ilustrativo. Si estuviésemos analizando un sistema de gestión universitaria, podríamos querer modelar que una persona tiene un nombre, un DNI y una dirección. Además un estudiante está matriculado

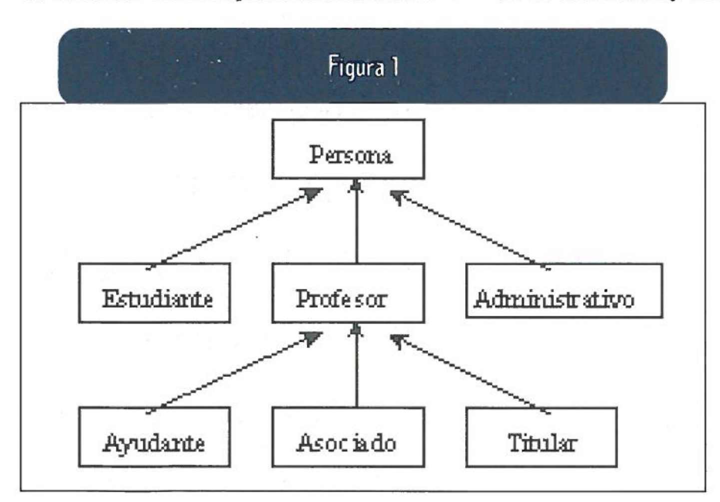
de un conjunto de asignaturas y un profesor tiene un sueldo e imparte también un conjunto de asignaturas. Vemos que los profesores y los estudiantes por ser personas comparten un conjunto de características, pero además cada uno tiene las suyas propias. Sería interesante definir primero las características de la persona y ser capaces de reutilizarlas posteriormente para definir los estudiantes y los profesores, a los que añadiríamos sus características propias.

La herencia es un mecanismo de abstracción que capta la relación *es un (is a)* entre dos clases de un sistema de información cuando se modela desde una perspectiva orientada a objetos.

En el ejemplo anterior diríamos que un Estudiante *es una* Persona. Estableciendo de esta manera una relación de herencia entre ambas clases. Conceptualmente, se dice que la clase Estudiante *hereda* de la clase Persona.

A las clases Estudiante y Profesor se les llama clases *especializadas, subclases o clases hijas*, y a la clase Persona *clase generalizada, base, superclase o clase padre*. Simplemente es cuestión de nomenclatura y dependiendo de la metodología que estemos consultando aparecerá un nombre u otro, pero en definitiva todos ellos recogen ese mismo concepto de relación *es un*. A lo largo del artículo se utilizará indistintamente cualquiera de ellos.

Figura 1



Es especialmente importante entender que un ejemplar de una clase hija no es un ejemplar diferente al de su clase padre, sino el mismo con características añadidas.

Si a la relación de herencia le añadimos la coletilla "simple" - relación de herencia simple - estamos diciendo que la clase hija no tiene ninguna otra clase padre. Por contra, si una clase hija tiene al menos dos clases padre estamos ante un caso de herencia múltiple. En el siguiente apartado del artículo se estudia en profundidad este nuevo tipo de relación de herencia.

Con la herencia, partiendo de una clase se obtiene una nueva que es una versión más "refinada" de la primera. Con esto se quiere decir que la clase especializada hereda todas las propiedades de su clase base y además puede añadirle otras nuevas e incluso modificar las heredadas. A simple vista, intuimos la potencia de la herencia como mecanismo de reutilización de partes del análisis ya elaboradas. Posteriormente, veremos que en implementación sucede lo mismo, es decir, que la clase hija hereda los atributos y métodos programados en la clase padre, teniendo así una eficaz reutilización de código.

Una vez conocido el concepto de herencia, vamos a introducir un grado más de complejidad viendo si es posible representar más de un nivel de herencia de forma simultánea. Una primera regla a tener en cuenta es que cualquier clase puede tener tantas clases hijas como sea necesario, sin existir limitación en cuanto al número. Además, cada una de estas clases hijas puede tener sus propias clases hijas; obtendríamos así clases en la jerarquía que son a la vez padres e hijas. Aplicando las dos reglas anteriores se pueden obtener redes de herencia con las que modelar sistemas de información de gran complejidad.

(1) Con las metodologías OO los análisis se realizan en tres fases [Rum95]: Modelo de Objetos, Modelos Dinámico y Modelo Funcional.
(2) El ejemplo propuesto aparecería representado en la *figura 1*.

La herencia es un mecanismo de abstracción que capta la relación es un (is a) entre dos clases de un sistema de información

Representación de la herencia

La gran mayoría de las metodologías - aunque como veremos más adelante OMT no lo hace así - representan la relación de herencia uniendo con una flecha de trazo continuo las clases relacionadas. En el extremo superior de la flecha aparece la clase padre y en el inferior la clase hija. De esta forma son fácilmente identificables las relaciones de herencia establecidas en el "modelo de objetos"(1) de un análisis.

Examinando ahora el gráfico obtenido podemos concluir que las relaciones de herencia quedan representadas por el concepto matemático de árbol (interpretando que las clases son los nodos y las relaciones los arcos). En una jerarquía de

herencia simple, por lo tanto, siempre habrá una única clase en lo más alto y además no existirán ciclos.

Los Roles

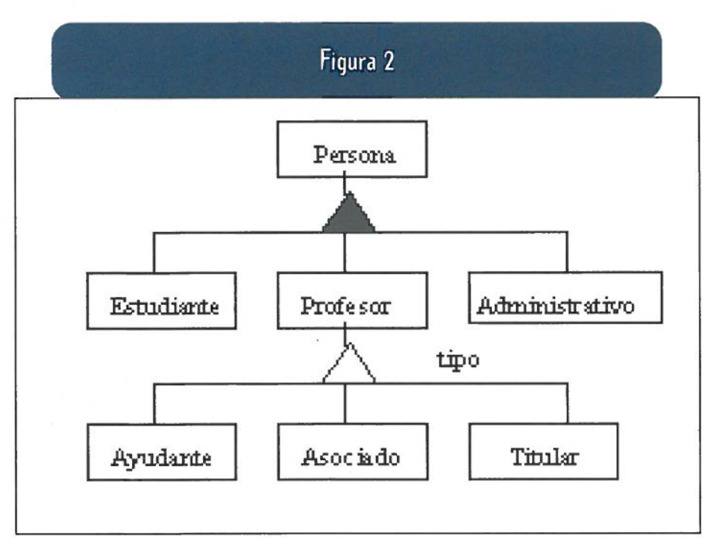
El concepto de rol en la herencia introduce la temporalidad en la relación entre una clase hija y una padre. Se dice que un ejemplar de la clase padre juega el papel (rol) de una clase hija durante un cierto periodo de tiempo, tras el cual esa faceta del ejemplar queda destruida. Destacar el hecho de que no son dos objetos distintos el padre y el hijo, sino el mismo con varias facetas que pueden ser o no simultáneas.

En el ejemplo, no expresamos si un objeto Estudiante lo será siempre o por el contrario puede dejar de serlo y permanecer en el sistema con la faceta de Profesor o Administrativo.

En general, con la introducción de la temporalidad en la herencia se plantean dos situaciones. La primera consiste en saber si un objeto de una clase padre es siempre obligatoriamente también un objeto de alguna de sus clases hijas. Éste es el típico caso de la relación de herencia entre la clase Persona y las clases Hombre o Mujer, obligatoriamente todo objeto de la clase Persona debe ser hombre o mujer. Muchas metodologías llaman a este tipo de herencia *permanente*. La segunda, a la que se suele llamar *temporal o rol*, se da cuando en la relación de herencia, el ejemplar adopta determinadas facetas (correspondientes a las clases hijas) durante un periodo concreto. En el ejemplo, donde especializábamos Persona en Estudiante, Profesor o Administrativo, la persona puede tener activas varias facetas simultáneamente o no. Pero, la diferencia con la herencia permanente consiste en que modelamos que nuestra persona puede ser, y dejar de ser, Estudiante, Profesor y/o Administrativo, a lo largo de su vida.

OMT utiliza el término rol en la asociación, por tanto no tiene nada que ver con el concepto que acabamos de exponer y además no será tratado ya que el artículo se dedica exclusivamente a la herencia. Esta metodología sí que representa el hecho de tener varios hijos simultáneamente, como veremos.

Figura 2



Generalización

Dentro de la herencia es importante mencionar el concepto de *generalización*. Éste es el mismo que el concepto de relación *es un*, pero con otra visión distinta. La generalización o relación *como un (as a)* se entiende como la operación inversa de la especialización, es decir, hace énfasis en entender la relación de herencia mirándola desde la clase hija hacia la clase padre.

Básicamente la generalización consiste en identificar todas aquellas características que son comunes a un conjunto de clases y con ellas hacer emerger una nueva clase. La clase así obtenida se dirá generalizada de las anteriores. En el ejemplo anterior diríamos que una Persona es una generalización de Estudiante, Administrativo y Profesor, a la vez que éste es una generalización de Ayudante, Asociado y Titular.

El lector debe entender que conceptualmente se habla de la misma idea y que son las metodologías las que proporcionan nomenclaturas distintas. Dentro del mundo de los lenguajes de programación OO, se da soporte a la herencia únicamente desde el concepto de especialización.

La definición de los conceptos de *especialización* y/o *generalización* determina el modelo de herencia que una metodología OO soporta. En el artículo estudiaremos el modelo de herencia que OMT como metodología de análisis y diseño OO propone.

Modelo de herencia de OMT

OMT da soporte metodológico al concepto de herencia, representándola con un triángulo equilátero que conecta a la clase padre con todas sus clases hijas, veamos a continuación el modelo de herencia que propone la metodología.

La clase hija hereda de su clase padre tanto la estructura como el comportamiento, es decir, los atributos y los métodos. Sin poder eliminar ninguno de éstos. A la capacidad que tienen las subclases de añadir nuevos atributos y métodos OMT le llama *extensión*.

La clase especializada puede redefinir los métodos heredados (*overriding*). En general, las metodologías obligan a que la redefinición no cancele comportamiento, es decir, que haga todo lo que hacía como padre y además añada comportamiento como hijo. Pero OMT es más flexible en este aspecto y permite la redefinición completa del método, aunque éste debe mantener el *protocolo* ex-

terno (argumentos y valor de retorno). Además se permite a una subclase limitar los atributos de sus antecesores. Esto se conoce como *restricción*.

La metodología distingue dos tipos de herencia. Aquella cuyas clases especializadas son *disjuntas* y aquella que permite el *solapamiento* entre clases hijas. Esta última hace referencia al concepto de simultaneidad que explicábamos al tratar los roles. La generalización disjunta se representa con un triángulo hueco y la solapada con un triángulo negro. La figura 2 presenta el ejemplo anterior utilizando la metodología OMT.

La relación de herencia de la parte superior del gráfico expresa que una persona puede ser estudiante, profesor, o administrativo, o combinaciones de estudiantes, profesores y administrativos.

La herencia disjunta que aparece en la parte inferior del gráfico obliga a que un atributo de la superclase actúe como discriminador de las instancias de la clase. En el ejemplo, es el atributo tipo de profesor el que realiza esta función, siendo ejemplares de la clase Ayudante aquellos objetos que tienen el valor "categoría primera" en el atributo tipo de profesor, ejemplares de la clase Asociado los que su valor es "categoría segunda" y los de la clase Titular su valor será "categoría tercera".

Listado 1. Declaración de clases en Visual C++.

```
class Cpersona
{
protected :
        CString m_dni; //número de dni
        CString m_nombre;
        CString m_direccion;
//...

public :
        CPersona (); //constructor
        CPersona ( CString dni, CString nombre, CString direccion );
Virtual void calcularSalario();
};

class CProfesor : public CPersona
{
private :
                CString m_numCarnetProfesor; //número de carnet de profesor
                CStringList ml_asignaturas;       //lista de asignaturas en las
                                                   //que está matriculado
        public :
                CProfesor () ;//constructor
                CProfesor( CString dni, CString nombre, CString direccion, CString
                                numCarnet);
        void calcularSalario();
};
```

Ejemplo en Visual C++

En el listado 1 podemos ver las declaraciones en VisualC++ de las clases Persona y Estudiante que nos servirán para ilustrar una posible implementación de la herencia simple.

Los atributos de CPersona son declarados como "protected" para que puedan ser accedidos por las funciones de CProfesor . Si se hubiesen declarado "private" las funciones de CProfesor no los podrían utilizar, con lo cual es como si no hubiesen sido heredados.

Otro caso interesante es el del constructor. El constructor de la clase hija deberá llamar al constructor de su clase padre y proporcionarle los argumentos requeridos. Veamos la implementación del constructor de estudiante:

```
CProfesor : : CProfesor ( CString dni,
    CString nombre,
CString direccion, CString numCarnetProfesor)
: CPersona (dni, nombre, direccion ),
m_ numCarnetProfesor (numCarnetProfesor )
{
}
```

Visual C++ permite que una función declarada en una clase base pueda tener diferentes implementaciones en sus descendientes a través del concepto de *función virtual*. Una función virtual es aquella que puede ser redefinida en todas las clases hijas de aquella en la que esté declarada, pero obligatoriamente debe ser definida en ésta. Para las clases en las que no se haga una redefinición explícita, la implementa-

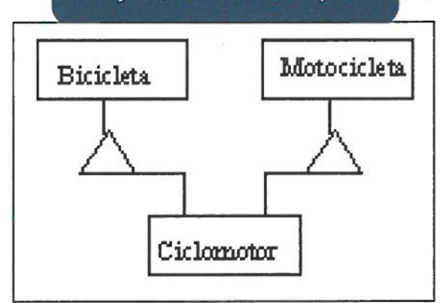
ción válida es la realizada en la clase antecesora más cercana en la red de herencia. En el ejemplo, la función calcularSalario() podrá tener diferentes implementaciones en las clases Profesor y Administrativo.

Ejemplo en Delphi

Para el caso Delphi las clases se definen como en el listado 2.

Podemos ver en el listado 2 cómo se ha implementado el concepto de función virtual en Delphi. Además, fijándonos en el constructor del hijo, vemos cómo llamamos explícitamente para que nos inicialice los atributos del padre con los argumentos pertinentes, similar al ejemplo VisualC++ pero con distinta sintaxis.

Figura 3. Herencia múltiple.




```
constructor TProfesor.Create(dni: string;
        nombre:string;
direccion:string; numCarnet:string);
begin
        inherited Create(dni,nombre,direccion);
        ménumCarnetProfesor:=numCarnet;
end;
```

OMT da soporte metodológico al concepto de herencia, representándola con un triángulo equilátero que conecta a la clase padre con todas sus clases hijas

Listado 2. Definición de clases en Delphi.

```
TPersona= class
  protected
      m_Dni: string;
      m_Nombre: string;
      m_Direccion: string;
      //...
   public
      constructor Create(dni:string;nombre: string; direccion:string);
      destructor Destroy;
      procedure CalcularSalario; virtual ; // indica que puede ser redefinido
      //...
  end;
TProfesor= class (TPersona)
   private
      m_numCarnetProfesor: string;
      ml_Asignaturas: string;
   public
      constructor Create(dni: string; nombre:string; direccion:string; numCarnet:string);
      destructor Destroy;
      procedure CalcularSalario; override; // indica que un método heredado se redefine
      // ...
  end;
```

Herencia múltiple

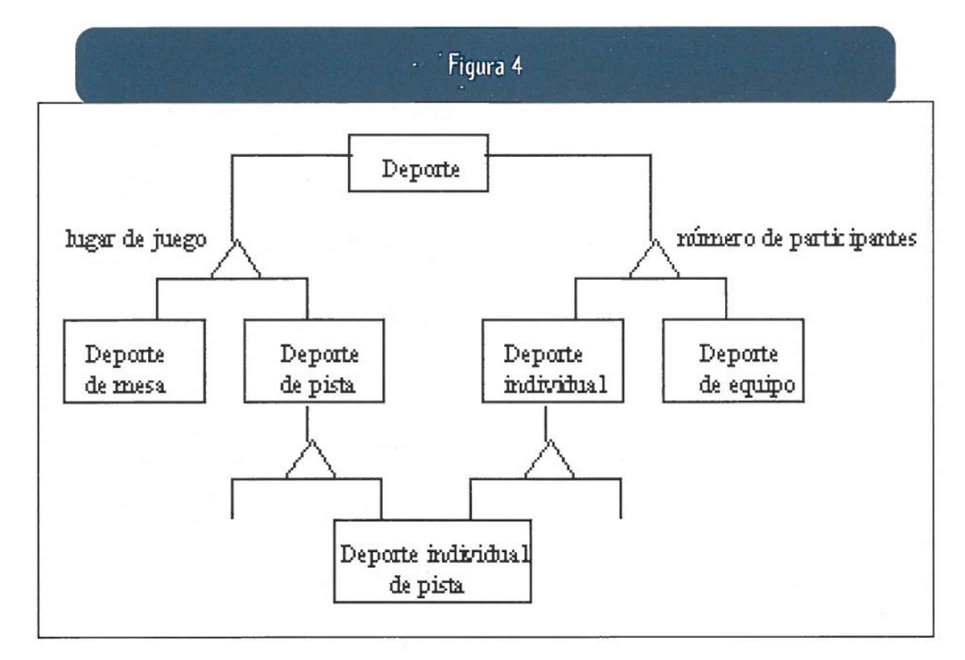
En todos los ejemplos que hasta ahora hemos visto sucedía que cuando una clase era especializada tenía únicamente un padre. Cuando nos encontramos ante una situación en la que una clase especializada tiene dos o más padres decimos que esa clase es *herencia múltiple* de sus clases padre.

El ejemplo de herencia múltiple (HM) de la figura 3 - representado con la notación de OMT - nos ayudará a verla con más claridad.

El ejemplo expresa que un ciclomotor es una bicicleta a la vez que una motocicleta y por tanto tendrá características de ambas clases. Podemos decir que el ciclomotor hereda de la motocicleta propiedades tales como tener depósito para gasolina, acelerador, etc., y de la bicicleta los pedales...

Con la HM se consigue que una clase herede características de más de un padre,

Figura 4



haciendo así más potentes los mecanismos de reutilización, pero también tiene en su contra que hace que los sistemas pierdan su sencillez conceptual y las implementaciones adquieran una notoria complejidad.

Las jerarquías de herencia no quedan representadas ahora por árboles sino por grafos acíclicos, ya que cualquier nodo de la jerarquía puede tener más de un padre. Esto incide también en la pérdida de sencillez conceptual a la que nos referíamos, ya que el concepto de árbol y sus algoritmos de recorrido están más próximos a la forma de pensar de las personas que el concepto de grafo.

Pero los problemas de la HM no acaban aquí, también debe evitarse el hecho de que una clase pueda heredarse más de una vez en la jerarquía, así como heredar la misma propiedad de varios padres.

Todos estos problemas han llevado a muchos lenguajes de programación OO a prescindir de ella como por ejemplo Delphi o Java.



OMT

OMT como metodología de análisis permite la existencia de clases con HM y las llama *clases unión*. Una clase puede tener HM dentro de una generalización con solapamiento, pero nunca la podrá tener dentro de una generalización disjunta, esto es lógico ya que si las subclases son disjuntas nunca habrá objetos en la intersección de ambas y por tanto no podrá emerger una nueva clase. No habrá ningún problema en crear una clase unión procedente de dos generalizaciones distintas.

Como decíamos, muchos lenguajes de implementación OO carecen de HM, en ellos programar las clases unión puede resultar difícil. La metodología es consciente de este problema y propone algunos "trucos" para eliminarla en fase de análisis.

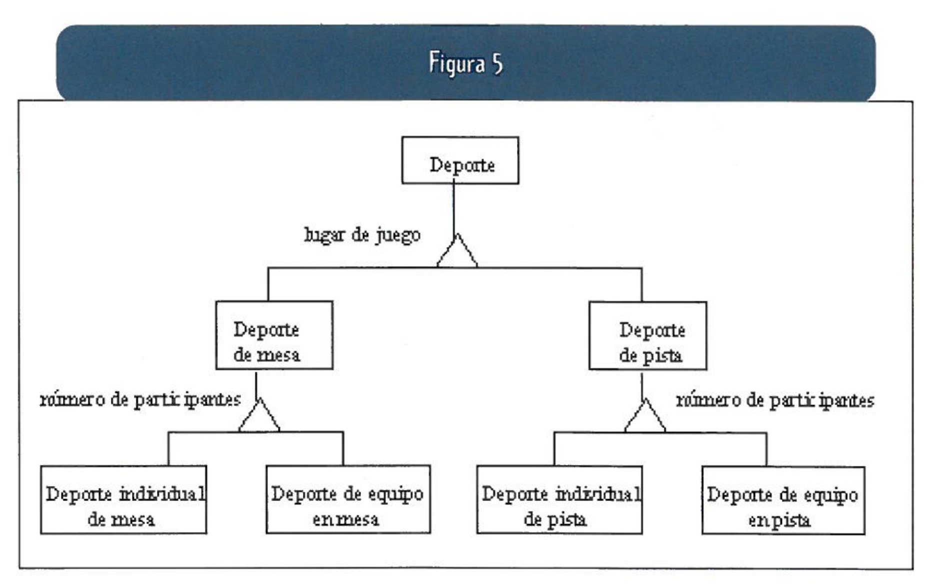
Veamos ahora un ejemplo (figuras 4, 5 y 6) que nos servirá para explicar estos trucos. Los "deportes" según su lugar de juego se pueden generalizar disjuntamente en "deportes de mesa" o "deportes de pista", y según el número de participantes pueden ser "deportes individuales" o "deportes de equipo", ésta también es una generalización disjunta. Por HM podemos obtener la clase que representan aquellos deportes que siendo individuales son practicados en una pista, naciendo así la *clase unión* "deporte individual de pista". Aunque esta clase procede de dos generalizaciones distintas, la metodología no se opone a su creación.

Trucos

1) Generalización anidada.

Este truco consiste en factorizar por una generalización y después por otra. Observando el ejemplo, vemos cómo la generalización del número de participantes original la hemos trasladado debajo de la generalización por lugar de juego. Esta aproximación tiene la desventaja de multiplicar las combinaciones, ade-

Figura 5



más de violar el espíritu del modelo OO, por lo que no la consideramos muy recomendable.

2)Utilizando la agregación.

Esta aproximación utiliza el concepto de *delegación*, que consiste en que un objeto pasa una operación a otro objeto para que la realice. Consiste en convertir alguna o todas las generalizaciones en una agregación de la superclase. De esta forma no tenemos un único ejemplar de Deporte sino dos. Deporte delegará en "Deporte según participantes" aquellos métodos de sus clases hijas.

Desde nuestro punto de vista estas soluciones o trucos que propone OMT son el resultado de adaptar el análisis original a una solución de implementación, sin embargo para ser fieles al paradigma OO las únicas limitaciones que debería aceptar un análisis OO son las propias de la metodología. En otras palabras, el análisis auténtico es el que contiene la HM, si así lo hemos captado al estudiar el problema.

Ejemplo en VisualC++

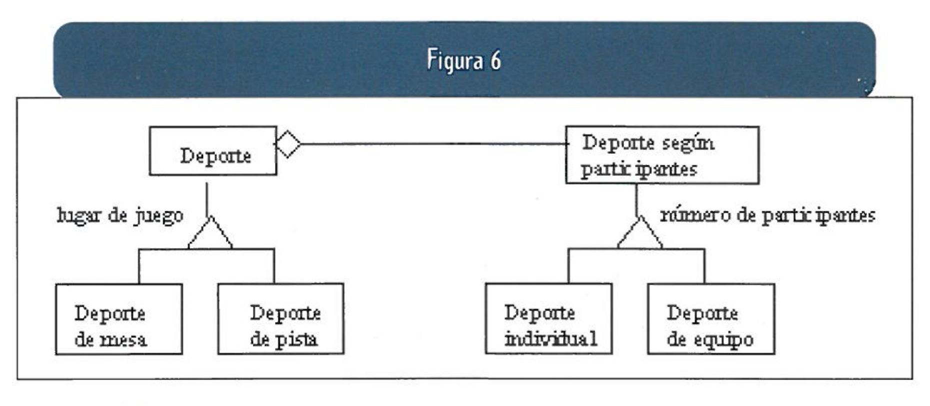
La implementación en VisualC++ correspondiente a las clases "Deporte de pista" y "Deporte individual de pista" sería la que aparece en el listado 3.

Haciendo virtuales las clases base conseguimos que el mismo objeto de la clase Deporte sea utilizado para representar todas las ocurrencias en las clases especializadas. Evitamos así el problema que habíamos comentado anteriormente de que una clase herede de otra antecesora más de una vez.

```
void CDeportePista::Setcubierta( BOOL valor )
{
        m_cubierta= valor;
}

BOOL CDeportePista::estacubierta()
{
        return m_cubierta;
}
```

Figura 6



Las funciones que se pueden definir y podrían ser posteriormente implementadas en la clase CDeportePista definida en el ejemplo anterior se hacen visibles en la clase llamada CDeporteIndividualPista.

Simulación de la herencia múltiple en Delphi

Cuando nos encontramos inmersos en el nivel de la implementación, programando con un LPOO que carezca de HM, deberemos ser capaces de sacar partido de todas las posibilidades que éste nos proporciona y que no son inherentes al modelo objetual. Como ejem-

Listado 3

```
class CDeportePista: public virtual CDeporte
{
protected:
BOOL m_cubierta;        //TRUE si el deporte se juega en
                        //pista cubierta.
        public:
                        CDeportePista() ;//constructor

protected:
void Setcubierta( BOOL valor );
        BOOL estacubierta();
};

class CDeporteIndividualPista: public virtual CDeportePista
{
        public :
                        CDeporteIndividualPista() ;//constructor
        //...
};
```

Listado 4

```
TDeporte_Pista= class
cubierta: Boolean;
...
function estacubierta:Boolean;
end;
TDep_individual_pista= class
...
FDeporte_Pista: TDeporte_Pista;
FDeporte_Individ: TDeporte_Individual;
...
function Getcubierta:Boolean;
procedure Setcubierta(const valor:Boolean);
property cubierta:Boolean read Getcubierta write Setcubierta;
function estacubierta:Boolean;
end;
function TDep_individual_pista.Getcubierta:Boolean;
begin
 Getcubierta:= Deporte_Pista.cubierta;
end;
procedure Dep_individual_pista.Setcubierta(const valor:Boolean);
begin
 Deporte_Pista.cubierta:= valor;
end;
function TDep_individual_pista.estacubierta:Boolean;
begin
 estacubierta:= Deporte_Pista.estacubierta;
end;
```

plo, vamos a ver cómo el lenguaje Delphi que no tiene HM la puede simular mediante unas características particulares llamadas *propiedades*.

Extraeremos del Listado 3 las clases Deporte de pista y Deporte Individual de Pista para plasmar estas ideas, viendo el código para simular en la *clase unión* los atributos y métodos de una de las clases padre.

La idea consiste en tener en la *clase unión* un puntero a cada padre. Los atributos de un padre se redefinirán como propiedades Delphi en la hija y accederán al valor original a través del puntero a su padre correspondiente. Los métodos de un padre se redefinirán también como métodos en la hija que llaman al método original del padre correspondiente a través del puntero. Ver listado 4.

La propiedad "cubierta" en la hija TDep_individual_pista llama al atributo original de TDeporte_Pista mediante las funciones de lectura/escritura (Getcubierta/Setcubierta respectivamente) asociadas a dicha propiedad.

El método "estacubierta" en la hija TDep_individual_pista llama al método original de TDeporte_Pista directamente a través del puntero existente en TDep_individual_pista.

Esto pretende ser sólo la idea de la implementación puesto que al adoptar esta implementación entramos en los problemas de ambigüedades que plantea la herencia múltiple.

Clases Abstractas

En el modelo orientado a objetos una *clase abstracta* es aquella que no posee ejemplares en el sistema, ya que representa un concepto abstracto.

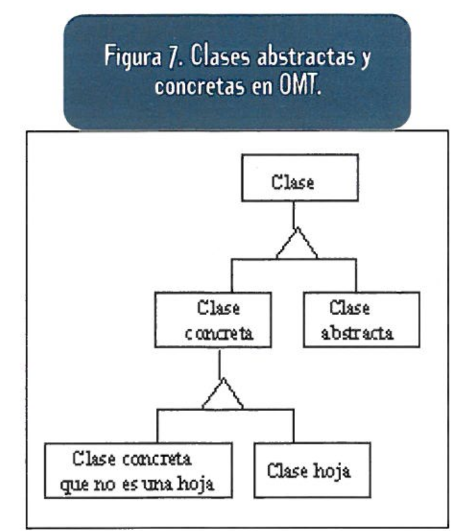
Un ejemplo de clase abstracta podría ser la clase "forma geométrica". Las formas geométricas pueden ser triángulos, rectángulos, etc. , pero nunca tendremos en el sistema ejemplares concretos de la clase "forma geométrica", por tanto ésta es abstracta, pero sí tendremos ejemplares de triángulos, rectángulos, y demás formas geométricas.

OMT

Veamos ahora el tratamiento que OMT da al concepto objetual de clase abstracta. La metodología distingue entre clases abstractas y *clases concretas*. Las primeras se ajustan a la definición general que dábamos para el modelo OO, y las segundas son aquellas que sí tienen ejemplares en el sistema.

Tanto las clases concretas como las clases abstractas pueden especializarse de clases concretas o abstractas. Ahora bien, se debe cumplir una regla, *"sólo las clases concretas pueden ser hojas en un árbol de herencia"*. De la regla anterior se

Figura 7. Clases abstractas y concretas en OMT.



deduce que una clase abstracta sólo tiene sentido si va a ser utilizada como clase base de alguna clase concreta, de otro modo sería una hoja del árbol y no se satisfaría la restricción.

La figura 7 muestra la jerarquía de herencia del modelo de objetos que define las clases abstractas y concretas en OMT.

La representación gráfica en OMT de las clases abstractas utiliza la misma notación que la herencia, de esta forma, el ejemplo anterior de las formas geométricas quedaría representado como muestra la figura 8.

Las clases abstractas tienen *operaciones abstractas*. Una operación abstracta es aquella en la que se proporciona el *protocolo* de ésta pero no su implementación. La utilidad de una operación abstracta es clara, la clase abstracta facilitará el *protocolo* de la operación, que será común a todas las clases hijas, y serán éstas las que realicen las implementaciones de acuerdo a sus necesidades, así se consigue una interfaz común para todas las clases hijas.

En el ejemplo, un caso de operación abstracta sería la operación "dibujar forma geométrica". La función se declararía en la clase "forma geométrica" y las implementaciones serían realizadas por las clases "triángulo" y "rectángulo".

Las clases concretas por lógica no pueden tener operaciones abstractas, ya que si un ejemplar de la clase intentase realizar una operación que no estuviese definida provocaría un error.

Vistos los conceptos que OMT proporciona relativos a las clases abstractas, centrémonos en las implementaciones que Visual C++ y Delphi ofrecen para trabajar con clases abstractas.

Ejemplo en VisualC++

En VisualC++ una clase abstracta es aquella que tiene una o más funciones *puramente virtuales*. Una *función puramente virtual* es una función virtual para la cual la clase no es capaz de proporcionar ninguna implementación. Ésta deberá ser realizada por las clases hijas. La forma sintáctica de convertir una función en puramente virtual es igualándola a cero en su declaración.

Veamos las implementaciones en VisualC++ correspondientes al ejemplo anterior:

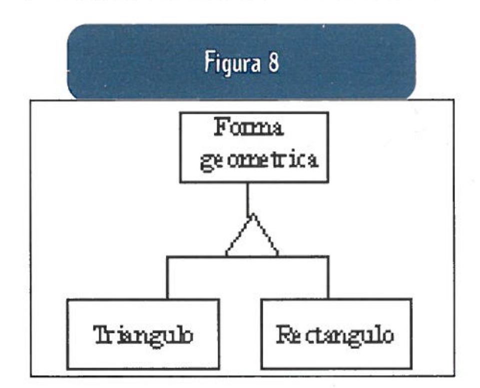
```
class CFormaGeometrica : public CObject
{
public:
        //función puramente virtual
        virtual void dibujar()= 0;
....
};
```

Al hacer la función dibujar de la clase CFormaGeometrica puramente virtual la estamos convirtiendo en abstracta. Por supuesto no se podrán declarar variables de ella, ya que provocaría un error en compilación.

```
CFormaGeometrica forma ; //error de compilación
```

La clase CTriangulo, aunque hereda de CFormaGeometrica, ya no es abstracta y sí se podrán declarar variables de ella. Además la clase CTriangulo deberá redefir la operación dibujar, en otro caso se produciría un error de compilación. Ver el listado 5.

Figura 8



Ejemplo en Delphi

En el caso de Delphi definiríamos la clase de las formas geométricas:

```
TFormaGeometrica= class
public
procedure Dibujar; virtual ; abstract;
//...
end;
```

Se puede observar cómo las *funciones puramente virtuales* (métodos abstractos) se marcarán en la clase padre mediante la palabra "abstract", además de las palabras "virtual" (en el padre) y "override" en el hijo que como vimos definen una *función virtual*.

A diferencia de Visual C++, Delphi permitiría declarar y crear un objeto TFormaGeometrica a pesar de tener una *función puramente virtual*.

```
var
 FormaGeometrica: TFormaGeometrica;
//...
FormaGeometrica:=TFormaGeometrica.Create;
//...
```

El problema surgirá si intentamos invocar un método abstracto, porque nos daría una excepción (error de ejecución).

Viendo cómo queda en Delphi la clase de los triángulos para completar el ejemplo:

Listado 5

```
class CTriangulo: public CFormaGeometrica
{
private:
int m_angulo1 ; //grados de uno de los ángulos
int m_angulo2 ; //grados de uno de los ángulos
int m_angulo3 ; //grados de uno de los ángulos

public:
          void dibujar(); //reescribe a CTriangulo::dibujar()

CTriangulo( int lado1, int lado2, int lado3 ) ;//constructor
~CTriangulo() ;//destructor
};


void CTriangulo::dibujar()
{
          //algoritmos para el dibujo de triángulos
}

CTriangulo triangulo ( 30, 30, 30 ) ;//correcto
```

```
TTriangulo= class (TFormaGeometrica)
 private
  m_angulo1: integer;
  m_angulo2: integer;
  m_angulo3: integer;
 public
  procedure Dibujar; override;
  constructor Create(lado1,lado2,lado3:integer);
  destructor Destroy;
  //...
 end;

procedure TTriangulo.Dibujar;
begin
//algoritmos para el dibujo de triángulos
end;

var Triangulo:TTriangulo;
```

Como conclusión diremos que las clases abstractas son un importante mecanismo dentro del modelo OO. Éstas son usadas para la reutilización de código que, como a lo largo del artículo venimos diciendo, es una de las características más importantes de la herencia. Otro uso importante es la capacidad que ofrecen de exponer una interfaz común a todas sus clases hijas sin aportar detalles de implementación.



Conclusiones

El modelo OO tiene un fundamento básicamente ontológico(2), por ello trata de recoger los mecanismos utilizados por las personas a la hora de pensar y darles una representación lo más natural posible dentro del modelo. La herencia es uno de los instrumentos más utilizados por los seres humanos en sus procesos de razonamiento, ésta es la razón por la cual adquiere un papel predominante en el paradigma de la orientación a objetos.

El artículo hace una revisión completa del *concepto* de herencia en los múltiples aspectos - herencia simple, herencia múltiple, ...- que ésta proporciona a la hora de modelar Sistemas de Información (SI) OO. Estos conceptos han sido abordados en sus fases de análisis y diseño con OMT, una de las metodologías hoy en vanguardia. La fase de implementación ha sido tratada utilizando dos de los lenguajes de programación OO más prestigiosos y utilizados por los programadores experimentados en OO.

Se ha demostrado que la utilización de toda la riqueza expresiva que la herencia aporta conduce a la obtención de SI sencillos de entender y altamente reutilizables. Ambas características son de extrema importancia a la hora de implementar sistemas, ya que conducen a la obtención de entornos fáciles de mantener y proclives a un crecimiento ordenado.

(2) La ontología es la rama de la filosofía que estudia cómo son las cosas en el mundo.

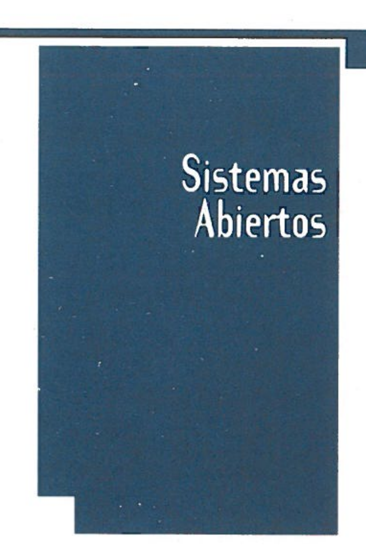
Bibliografía

• [Rum95] "Modelado y diseño orientados a objetos. Metodología OMT". James Rumbaugh et al. Prentice Hall, 1995.

• [Str93] "The C++ programming language". Bjarne Stroustrup. Addison-Wesley, 1993.

• [Kel95] "Delphi, a developerís guide". Vince Kellen. M&T Books (MIS:Press Inc.) 1995.

Creación de una intranet con LINUX



Carlos Àlvaro

Continuamos creando la intranet, configurando esta vez nuestros sistemas para que formen una red TCP/IP.

TCP/IP es una familia de protocolos que ha estado, desde sus orígenes, estrechamente ligada al mundo UNIX. Por lo tanto, no es extraño que sea este sistema operativo el que haya marcado la pauta del cómo configurar las conexiones y del dónde deben ir los ficheros relacionados con este tipo de comunicaciones. Otros sistemas operativos han aplicado estas estructuras haciéndolas compatibles con las suyas.

En el número 32 de "Sólo Programadores" decíamos que Linux era un clónico del sistema UNIX explicando lo que esto significaba, es decir, que había sido reescrito desde el principio siguiendo todos los estándares. Debido a esta fidelidad al estándar, a todos los efectos y no sólo en el ámbito de las comunicaciones, consideramos que Linux es un UNIX.

A continuación veremos cómo configurar nuestros sistemas para formar entre ellos una red TCP/IP. La propia instalación y configuración de Linux resuelve la aparición de los primeros servicios Internet.

La conexión física

Suponemos que tenemos nuestras dos máquinas con el sistema operativo instalado y sus respectivos interfaces de red (que, para nuestra intranet, serán dos tarjetas ethernet) unidas mediante cable coaxial ethernet. Ésta es la configuración más sencilla. Dejamos a la elección y a las posibilidades de nuestros lectores el considerar la utilización de más máquinas, o de otro tipo de cableados, como PDS directo o PDS sobre HUB's, etc.

En el arranque de ambos sistemas, Linux ha debido reconocer automáticamente la presencia de las tarjetas ethernet. En los mensajes que aparecen al iniciar el sistema, uno de ellos debe hacer referencia a ella, con su interrupción y su puerto de entrada/salida correctos, su zona de memoria reservada, en el caso de que la tarjeta la use, y cualquier otra característica reflejada en el manual que debe acompañar al modelo de tarjeta ethernet que poseamos. Es fundamental, antes de comenzar con el siguiente paso, tener la tarjeta de red instalada y reconocida por Linux.

¿El sistema no reconoce la presencia del adaptador de red? Es preocupante, pero no desesperemos tan pronto. Algunas ideas para solucionarlo pasan por:

- Comprobar que la tarjeta está correctamente pinchada en el slot correspondiente.

- Asegurarnos de que no existen otros dispositivos en el ordenador asignados a la misma interrupción o al mismo puerto.

Un dominio no es más que una agrupación de máquinas, a nivel lógico, con el fin de compartir los recursos de una red

Verificar, en el caso de los ordenadores con arquitectura PCI, que la bios no está interfiriendo la IRQ de nuestra tarjeta con las que reserva para los slots PCI.

De cualquier forma, siguiendo las instrucciones del manual de la tarjeta, intentar configurarla en otra interrupción y otro puerto.

Pasarle los programas de test del disquete que acompañaba a la tarjeta cuando la adquirimos.

Probar con otros softwares de otros sistemas operativos para asegurarnos de que no se trata de material defectuoso.

Buscar sobre la Internet relaciones entre el binomio que forma el modelo y fabricante de nuestra tarjeta con "Linux", (podemos encontrar instrucciones, drivers, parches o recomendaciones que nos puedan ser de utilidad).

En fin, hay muchas cosas que probar antes de tomar la decisión de obtener otra tarjeta distinta que, afortunadamente, tampoco es una decisión excesivamente cara.

Configuración de la pila TCP/IP

Diversos UNIX incluyen entre sus utilidades algunos programas y herramientas para la configuración de la pila de protocolos TCP/IP. En Linux, la utilidad que realiza esta tarea se llama 'netconfig'. Vamos a configurar de golpe nuestras dos máquinas y después explicaremos, paso por paso, el porqué de los valores introducidos.

Para rellenar los campos que esta utilidad irá solicitando hemos elegido ciertos valores que tomaremos como referencia. Por supuesto, es elección de cada uno cosas tales como el nombre de host, el nombre de dominio, etc.

Suponiendo que todo funciona sin problemas, entraremos en ambos ordenadores como "root" y ejecutaremos 'netconfig'. Las pantallas que aparecen a continuación (ver figuras 1 y 2) preguntan por una serie de valores que serán los siguientes para la máquina que hemos elegido como servidor:

Figura 1

```
Enter hostname: andora
Enter domain name for andora: intranat
Do you plan to ONLY use loopback? No
Enter IP address for andora
(aaa.bbb.ccc.ddd): 192.168.1.4
Enter gateway address
(aaa.bbb.ccc.ddd): 192.168.1.4
Enter netmask (aaa.bbb.ccc.ddd):
255.255.255.0
Will you be accessing a nameserver?
No
```

Y los siguientes para la máquina cliente aparecen en la figura 2.

Figura 2

```
Enter hostname: menteuse
Enter domain name for menteuse: intranat
Do you plan to ONLY use loopback? No
Enter IP address for menteuse
(aaa.bbb.ccc.ddd): 192.168.1.11
Enter gateway address
(aaa.bbb.ccc.ddd): 192.168.1.4
Enter netmask (aaa.bbb.ccc.ddd):
255.255.255.0
Will you be accessing a nameserver?
No
```

Significado de los campos

Se suele decir que el campo "hostname" es el nombre con el que es conocido nuestro ordenador, pero esto no es cierto del todo. Mejor expresado, "hostname" es el nombre de nuestro interface de red, ya que, si tuviéramos más de una tarjeta, habría que utilizar nombres y direcciones distintas para cada una de ellas. Nosotros, como suponemos que sólo tenemos una tarjeta de red para cada uno de los equipos, podemos hacer coincidir el campo "hostname" con el nombre de nuestro sistema. Nuestra máquina servidor se llamará 'andora', y la cliente, 'menteuse'.

El campo "domain name" es el nombre del dominio al que pertenece nuestra máquina. Un dominio no es más que una agrupación de máquinas, a nivel lógico, con el fin de compartir los recursos de una red. Para todas las máquinas integrantes de un dominio se establecen una serie de características comunes en lo referente a las comunicaciones, pudiendo determinarse un conjunto de derechos, permisos y restricciones comunes a ellas. El dominio establecido en nuestra intranet se llamará, en nuestro ejemplo, 'intranat'.

Se denomina "loopback" a la red que la pila TCP/IP forma internamente consigo misma en una máquina determinada. Una máquina ha de ser cliente y servidor de sí misma. Esto es suficiente para poder utilizar TCP/IP de forma interna pero, puesto que nosotros queremos establecer comunicaciones con otros ordenadores,

no nos limitaremos a utilizar el *loopback*. Todo ordenador que quiere entrar en red TCP/IP a través del loopback, lo hace mediante la dirección 127.0.0.1.

De esto se deduce que, en realidad, en nuestra máquina siempre vamos a tener dos redes: una será el *loopback* y la otra, la formada con el resto de ordenadores a través del cableado ethernet.

La dirección IP será la asignada a nuestra interfaz de red. Mediante la dirección se podrán capturar a través de la tarjeta de red los paquetes IP de los que somos destinatarios, saber quién los envía y permitir a los elementos de routing que sean capaces de hacerlos llegar correctamente a su destino.

Cuando netconfig solicita la dirección de gateway se está refiriendo al concepto de "gateway por defecto". Es decir, la dirección de una máquina que hace

Todo ordenador que quiere entrar en red TCP/IP a través del loopback, lo hace mediante la dirección 127.0.0.1

de acceso entre dos redes conectadas y a la que se envían los paquetes que, por tener una dirección IP que no pertenece a nuestra red no son para ninguno de los ordenadores directamente conectados a nosotros. Toda máquina sabe qué rango de direcciones pertenecen a su red y cuáles no mediante operaciones lógicas entre su dirección IP y la máscara de subred. Como nosotros no entraremos durante una temporada en las complejidades del routing, podemos establecer que la dirección del gateway por defecto es la de la interfaz en andora.

El 'netmask' o máscara de subred, aplicado sobre la dirección IP, "oculta" la dirección de red (que en nuestro caso es 192.168.1.0) y deja ver la dirección del host (4, en el caso de andora, y 11 en el caso de menteuse). Con esto se sabe qué paquete ha de ser puesto tal cual en la propia red y qué paquete ha de ser encapsulado y enviado al gateway por defecto, puesto que está destinado a un ordenador de otra red.

Un 'nameserver' o servidor de nombres es un servidor que mantiene unas tablas que relacionan el nombre de diferentes máquinas con sus correspondientes direcciones IP, de tal forma que recibe el nombre de la máquina con la que se quiere establecer una conexión y devuelve su dirección IP. El uso de servidores de nombre mejora grandemente las tareas de mantenimiento en redes grandes y evita multitud de errores debidos a la mala gestión de los puestos, las informaciones contradictorias en el direccionamiento, etc. Sin embargo, para nuestra intranet de dos máquinas no lo vamos a utilizar... de momento.

Comprobando que estamos en red

Después de esto, con todos los valores correctamente establecidos y ya desde el prompt del sistema operativo en andora, mediante la orden:

```
ping 127.0.0.1
```

comprobaremos la conexión interna de andora con su propio *loopback*, es decir, con la red TCP/IP que todo ordenador forma consigo mismo.

La orden:

```
ping 192.168.1.4
```

comprueba la conexión de andora consigo mismo a través de la tarjeta de red, y la orden: ping 192.168.1.11

comprueba la conexión de andora con menteuse.

Igualmente, desde la consola de menteuse, podemos introducir:

```
ping 127.0.0.1
ping 192.168.1.11
ping 192.168.1.4
```

para realizar las mismas comprobaciones.

La utilidad 'ping' envía paquetes ICMP ECHO_REQUEST a los hosts de la red, es decir, utiliza el mandato ECHO_REQUEST implementado en el protocolo de transporte ICMP (Internet Control Message Protocol), para generar una respuesta ECHO_RESPONSE del ordenador que recibe los paquetes. Con esto se comprueba que ambos ordenadores se encuentran en red y que es posible la comunicación entre ellos.

Si la instalación del sistema operativo ha acabado sin errores, las tarjetas no presentaban ningún problema, el cableado está bien conectado y no es defectuoso y se han introducido los anteriores valores en las pantallas de 'netconfig', no hay ninguna razón para que esto no funcione como esperamos.

Los servicios Internet

TCP/IP es un conjunto de protocolos desarrollado por una comunidad de investigadores en torno a la red ARPAnet, red militar americana ya desaparecida, con el objeto de compartir recursos a través de la misma. IP y TCP son dos de los protocolos de este grupo pero, al ser los más conocidos, la mayor parte de las veces se utiliza el término TCP/IP o IP/TCP para referirse a toda la familia.

En su estructura cliente/servidor, hay una clara distinción entre los programas que realizan una u otra función. Para cada servicio Internet existe un protocolo, a nivel se aplicación, optimizado para el tipo de labor que debe llevar a cabo; si bien es cierto que otros protocolos perte-

necientes a otras tareas podrían dar el mismo servicio.

Así, por ejemplo, es bastante común hoy en día la transferencia de ficheros utilizando el protocolo HTTP, en vez del protocolo FTP, que sería el más adecuado.

Tradicionalmente, los servicios más importantes proporcionados por TCP/IP son los siguientes:

- ### Transferencia de ficheros:

El protocolo de transferencia de ficheros FTP (*File Transfer Protocol*) permite a un usuario de una computadora recibir ficheros de otra computadora, o enviárselos. Los mecanismos de seguridad se manejan mediante la petición de un nombre de usuario y una password por parte del sistema remoto. En la implementación del protocolo deben estar contempladas las características propias de las diferentes máquinas y de los diferentes sistemas operativos, tales como el uso de juegos de caracteres diferentes o convenciones para señalar el fin de línea.

Obviamente, en el caso de que se realicen transacciones FTP orientadas a la actualización de ficheros, las modificaciones a los ficheros remotos se producen siempre en local, trabajando con una copia que luego será enviada.

- ### Login remoto:

El protocolo de terminal de red (TELNET) permite a un usuario abrir una sesión remota contra otro sistema de la red. Desde que comienza una sesión hasta que termina, los caracteres tecleados en el ordenador local son enviados directamente al otro ordenador. Las implementaciones más comunes de TELNET proporcionan generalmente emulaciones para los tipos más conocidos de terminal.

- ### Correo electrónico:

Permite el envío y recepción de mensajes con usuarios de otros ordenadores de la red. El modo más sencillo de implementar este servicio es manteniendo ficheros de correo (llamados "buzones" o "recipientes") en cada uno de los sistemas donde se pretende intercambiar mensajes con alguno de sus usuarios.

El sistema de correo electrónico es un método para añadir un mensaje a esos ficheros o recuperarlos. El problema principal aparece cuando los ordenadores entre los que se intercambia correo no tienen continuamente activo el gestor de correo, (bien por estar apagados o por no permitir la multitarea). Esto se solventa mediante la instalación de *mail servers*.

- ### *Network file systems:*

Permite a un sistema acceder a ficheros de otro ordenador dando la impresión de que son sistemas de ficheros locales. Las ventajas de este modo de trabajo son: el uso de unidades de gran capacidad por parte de varios ordenadores, la compartición de ficheros comunes de datos eliminando problemas de inconsistencias de la información y haciendo más fácil su administración, y los procesos de backup.

- ### Impresión remota:

Permite acceder a impresoras de otros ordenadores para compartirlas. RLP (Remote Lineprinter Protocol) es un ejemplo de protocolo orientado a este servicio.

- ### Ejecución remota:

Permite que los requerimientos de un programa puedan ser ejecutados en diferentes computadoras. Puede ir desde la sencillez de enviar un único comando al sistema remoto (RSH, REXEC), hasta la complejidad de elegir automáticamente aquel sistema que tiene la tasa más baja de ocupación de CPU en un determinado momento (*distributed shell* como RPC, *Remote Procedure Call*).

- ### Servidores de nombres:

Se utiliza en instalaciones de gran tamaño, donde hay que manejar y mantener una considerable cantidad de nombres (nombres de usuario y sus *passwords*, nombres y direcciones de computadoras, grupos, cuentas, etc).

- ### Servidores de terminales:

Se trata de pequeñas computadoras que sólo tienen que saber cómo se debe ejecutar TELNET (o cualquier otro protocolo de login remoto), aunque es conveniente que esté integrado con un servidor de nombres.

El servidor de terminales es el encargado de ofrecer conexiones entre los ordenadores que estén conectados a él, siendo capaz de ofrecer multiconexión, es decir, proporcionar intercambio de conexión a distintos servidores para un mismo cliente de una manera rápida y eficiente.

Algunos de los protocolos que hemos mencionados y otros que no han sido detallados (servidores de fecha, servidores de *who*, etc), han sido diseñados por distintos fabricantes y organizaciones como Sun o Berkley y no forman parte del conjunto oficial de protocolos de la Red de redes.

Conclusión

Al llegar a este punto, nos encontramos con nuestras dos máquinas completamente instaladas, configuradas para su trabajo en red y conectadas entre sí.

Como veremos en próximos números de "Sólo Programadores", estaríamos ya en disposición de utilizar algunos de los servicios Internet.

Otros están instalados pero no configurados, y los hay que todavía requieren de la presencia de ciertas piezas de software que habrá que añadir.

Automatización OLE

Juan Manuel Menéndez

En el artículo anterior intentamos sacar el máximo partido al Control Contenedor OLE, en este artículo vamos a subir un grado más en el control de las aplicaciones OLE, por medio de las posibilidades que nos suministra la Automatización OLE.

Automatización OLE

La Automatización OLE (vinculación e incrustación de objetos) es una interfaz que utilizan las aplicaciones para exponer parte de su código en forma de objetos OLE ante otros programas que actúan como clientes. Por ejemplo, una aplicación de hoja de cálculo puede exponer una hoja de cálculo o parte de la misma (celdas, intervalo de celdas) cada una de las cuales representa un tipo de objeto distinto.

Cuando una aplicación suministra sus servicios por medio de Automatización OLE, Visual Basic tiene acceso a los objetos que ésta expone a través del servidor.

A diferencia del control contenedor OLE que se vio en el artículo anterior, los métodos y propiedades a los que tenemos acceso por medio de está técnica son distintos para cada servidor, por lo que es necesario tener la información de la interfaz para cada uno de ellos, para lo cual habrá que recurrir a los manuales adecuados o al *Help* del servidor OLE con el que estemos tratando.

En este artículo crearemos un programa que requerirá los servicios de Excel y Word. El programa extraerá

Automatización OLE es el mecanismo por el cual podemos acceder a los métodos y propiedades de un servidor OLE

unos datos de una hoja Excel y se hará una fusión de datos con una plantilla de documento Word. Desde un punto de vista funcional vamos a utilizar el servidor Excel como "máquina" para calcular y el servidor Word como "máquina" para imprimir.

Para tratar con objetos de OLE automatización los primeros pasos a seguir son los siguientes:

- Definir las variables OLE Automatización.

Las técnicas de Automatización OLE son una de las más potentes herramientas de programación de que se dispone en Windows. Por medio de ellas podremos controlar los servidores OLE y ponerlos a nuestro servicio en las aplicaciones que desarrollemos.

Asignar a estas variables el objeto de OLE Automatización.

Existe en Visual Basic un tipo de datos para los objetos de OLE Automatización; este tipo de datos se denomina Object. La creación y definición de estos objetos se realiza de la siguiente forma:

```
Public Libro As Object
Public hSheetPrecios As Object
Public hSheetConsumos As Object
Public hSheetFacturacion As Object
```

Una vez definida la variable de objeto, tendremos que asignar la misma a un objeto. Visual Basic tiene dos funciones para crear objetos: si el objeto existe en un fichero en nuestro disco, deberemos recuperar el objeto (función GetObject), si el objeto no existe y lo vamos a crear, utilizaremos la función CreateObjetc. Ambas funciones devuelven un puntero de 32 bits a la instancia del objeto.

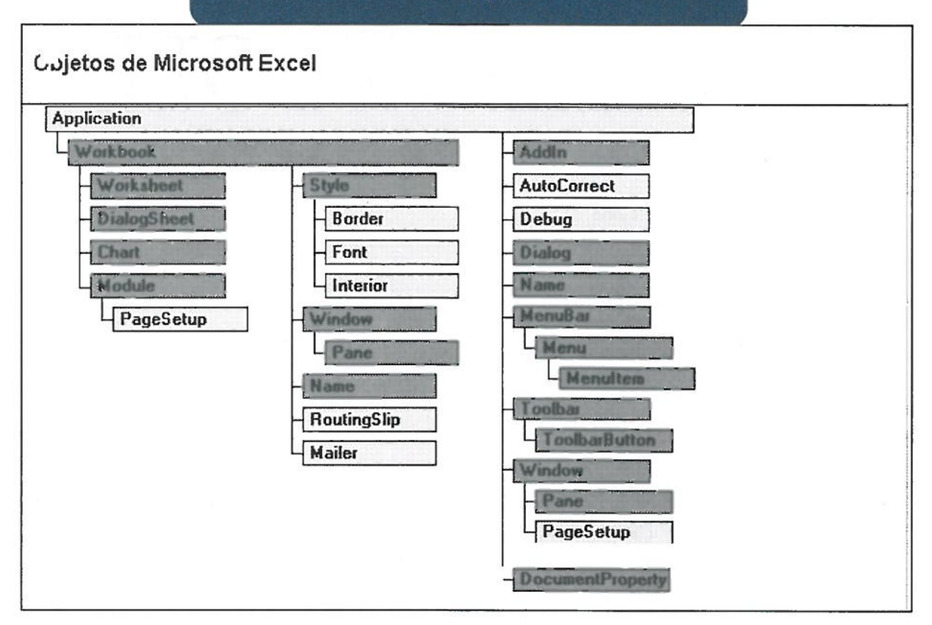
```
Set Hoja = CreateObject("ExcelSheet.5")
Set Hoja =
        GetObject("ruta_del_fichero","Excel.Sheet.5")
```

No es necesario especificar la clase del objeto si el sistema es capaz de identificar el tipo de objeto a partir del nombre del archivo, pues el sistema podría obtener la información del servidor OLE del

La clase del objeto no es necesario especificarla si el sistema es capaz de identificar el tipo de objeto a partir del nombre del archivo

Registry tal y como se vio en el primer artículo de la serie (nº 32 de Sólo Programadores). Cuando se crea el objeto con cualquiera de las dos funciones, se inicia la aplicación bien con documento inicial en blanco (caso de *CreateObject*), bien con documento existente (caso de *GetObject*).

FIGURA 1



El objeto ya está cargado en nuestra aplicación y antes de seguir es el momento de conocer algo sobre las características del mismo.

En la figura1 se puede ver la jerarquía de objetos que tiene Excel; cada objeto del gráfico tiene sus propios conjuntos de métodos y propiedades que deberemos conocer en profundidad para utilizar esta herramienta. Si no disponemos de los libros adecuados, podemos utilizar el excelente *Help* que nos viene con Excel. En concreto los capítulos:

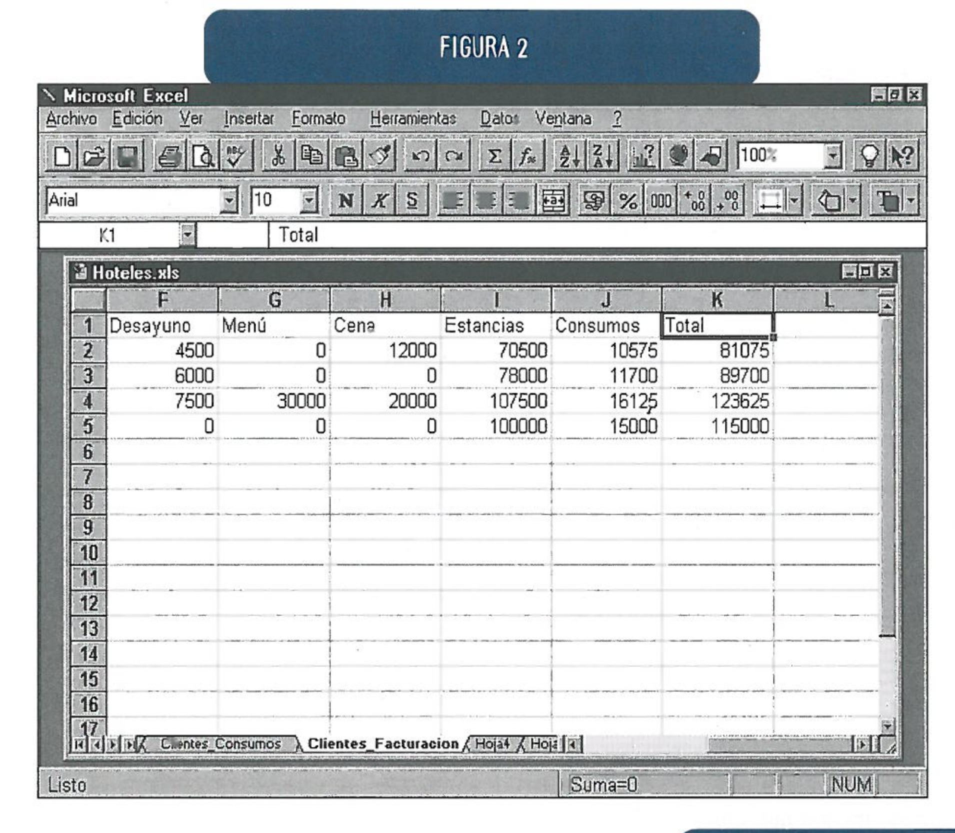
- Referencia a Microsoft Excel Visual Basic
- Referencia a Visual Basic para Microsoft Office
- Referencia a Visual Basic para Cuaderno Microsoft Office

Se puede encontrar una completa, aunque poco didáctica, información sobre este tema.

Un fichero de Excel (.xls) contiene un libro (*WorkBook*), que a su vez está dividido en *WorkSheets*. Cada *WorkSheet* está dividida en celdas(Cells), siendo cada celda la intersección de una columna con una fila. La hoja con que vamos a tratar tiene 4 *WorkSheet* a las que se ha dado nombre: Precios_1996, Clientes_Consumo, Clientes_Facturación. En la primera están los precios de los diversos elementos, en la hoja Clientes_Consumo aparece el consumo realizado por los clientes y en Clientes_Facturación (ver figura 2) podemos ver el importe de lo consumido. Las diversas celdas de esta hoja están calculadas con valores existentes en las hojas previas. El proceso que vamos a realizar pretende dar un ejemplo de las posibilidades de Excel con OLE Automatización. Utilizando la potencia de Excel para realizar los cálculos, podemos crear una hoja con los cálculos necesarios para introducir desde el programa los datos en determinada celda y obtener los resultados en otra celda. Las ventajas de este método son:

- Utilizamos la hoja de cálculo como máquina de operar.
- Flexibilidad, ya que cualquier variación en los cálculos y constantes únicamente afecta a las celdas, evi-

FIGURA 2



tando las modificaciones de código y compilaciones del mismo.

El primer paso es crear el objeto WorkBook de nuestro fichero, lo que realizamos con la sentencia:

```
Set Libro =
        Hoja.Application.Workbooks(nombreLibro)
```

Tenemos cargado nuestro libro en la variable de tipo Objeto Libro. Para acceder a las hojas del libro utilizaremos el método *WorkSheets*, que devuelve un objeto que representa una hoja de cálculo (un objeto *Worksheet*). La sintaxis de esta propiedad es:

```
Set hSheetFacturacion =
        Libro.WorkSheets(nombreSFacturacion)
```

El siguiente objetivo es obtener información de las celdas. Para ello utilizamos el método *Cells* del objeto *WorkSheet*; este método nos devuelve una referencia a un objeto que representa una celda de la hoja.

```
Set celdaCliente = hSheetFacturacion.Cells(i, 1)
```

Por fin obtenemos el valor de la celda por medio de la propiedad *Value* del objeto:

```
wstring = celdaCliente.Value
```

En el programa se ha ido a través de la jerarquía en distintos pasos con fines didácticos, pero en una codificación real se puede acceder directamente al objeto sin necesidad de ir asignando una variable para cada elemento en el árbol. Para hacerlo, se agrega al final del nombre del archivo una exclamación de cierre (!) así como una cadena que identifique la parte del archivo que se desea activar:

```
Set Hoja =
        GetObject("ruta_del_fichero"!nombreLibro!
        nombreSFacturacion)
```

Una vez finalizado nuestro trabajo con la hoja, la cerramos con el procedimiento *.Quit*. Si hemos modificado el contenido de la misma, deberemos grabarla previamente con el méto-

FIGURA 3

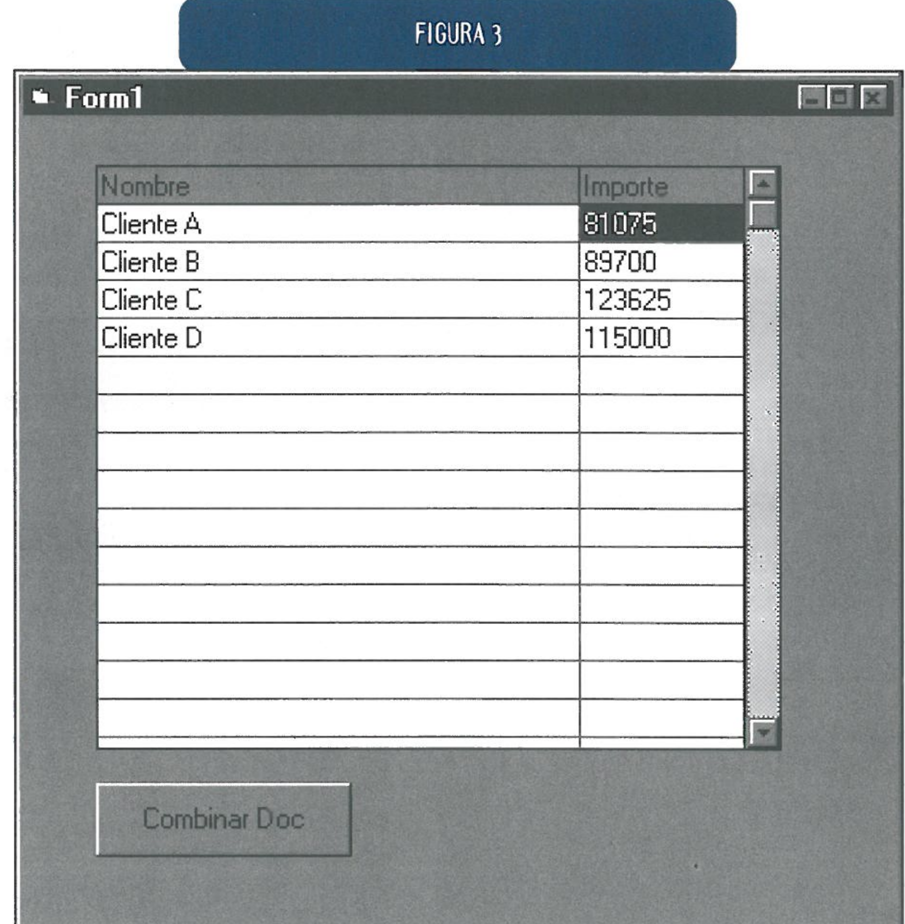


Tabla 1. Información de una combinación de documentos.

0 La ruta de acceso y el nombre de archivo de la fuente de datos

1 La ruta de acceso y el nombre de archivo de la fuente de registro

2 Un número, devuelto como texto, que indica cómo se suministran los datos en una operación de combinación de correspondencia:

- 0 Desde un documento de Word o a través de un archivo convertidor de Word
- 1 Intercambio Dinámico de Datos (DDE) desde Microsoft Access (sólo para Windows)
- 2 DDE desde Microsoft Excel
- 3 DDE desde Microsoft Query (sólo para Windows)
- 4 Open Database Connectivity (ODBC) (sólo para Windows)

3 Un número, devuelto como texto, que indica cómo se suministra la fuente de registro en una operación de combinación de correspondencia. Consulte los valores y las descripciones para el Tipo 2.

4 La cadena de conexión de la fuente de datos

5 La cadena de consulta (instrucción SQL)

do *.Save*, si no queremos que Excel le pida al usuario salvarla. Más información sobre este tema la veremos en el apartado "Algunas diferencias".

Como paso final después de haber realizado nuestro trabajo, deberemos liberar los recursos utilizados. Para ello asignamos el valor Nothing a cada variable de tipo Objeto que hayamos utilizado en nuestro programa:

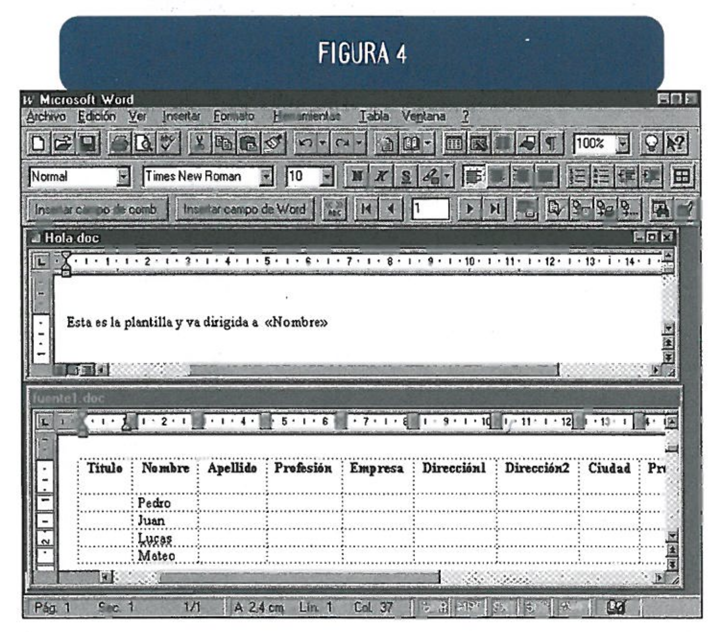
```
Set Hoja = Nothing
```

Ejecutemos el programa y podremos ver cómo los valores del control Grid se corresponden con las celdas de las columnas 1 y 11 de la Hoja "Clientes_Facturacion" de nuestro libro (Ver figuras 2 y 3).

Trabajando con Word

Dejemos la hoja de cálculo y pasemos a trabajar con otro servidor OLE, nada menos que con Word, y realizaremos una tarea algo más complicada que abrir un fichero e imprimirlo; realizaremos una fusión de documentos. Pero antes de seguir, explicaremos un poco en qué consiste la fusión de documentos (o combinar correspondencia). La fusión de documentos es una herramienta de los procesadores de texto, no solo de Word, mediante la cual podemos unir dos documentos, uno que llamaremos *plantilla* y otro que llamaremos *datos*. El documento *datos* contiene una tabla con campos a los que se da un nombre y el documento *plantilla* contiene un texto con formato y una referencia a los campos del documento *datos*. En la figura 4 podemos ver los dos documentos utilizados en la fusión y en la figura 5 el primer documento de la fusión.

FIGURA 4



Vamos a crear el código necesario para poder realizar la fusión desde nuestro programa apretando un botón del formulario.

El primer paso para poder trabajar con OLE Automatización es buscar la información necesaria acerca de nuestro servidor, como el ejemplo que vamos a tratar está lo más simplificado posible, nos vamos a ayudar, al igual que hemos hecho con Excel, del *Help* que viene con Word. El lector observará que la sintaxis de las sentencias que aparecen en la documentación del producto es sensiblemente diferente a la utilizada en el programa. En concreto en el *Help* de Word los parámetros de las instrucciones vienen con palabra de clave:

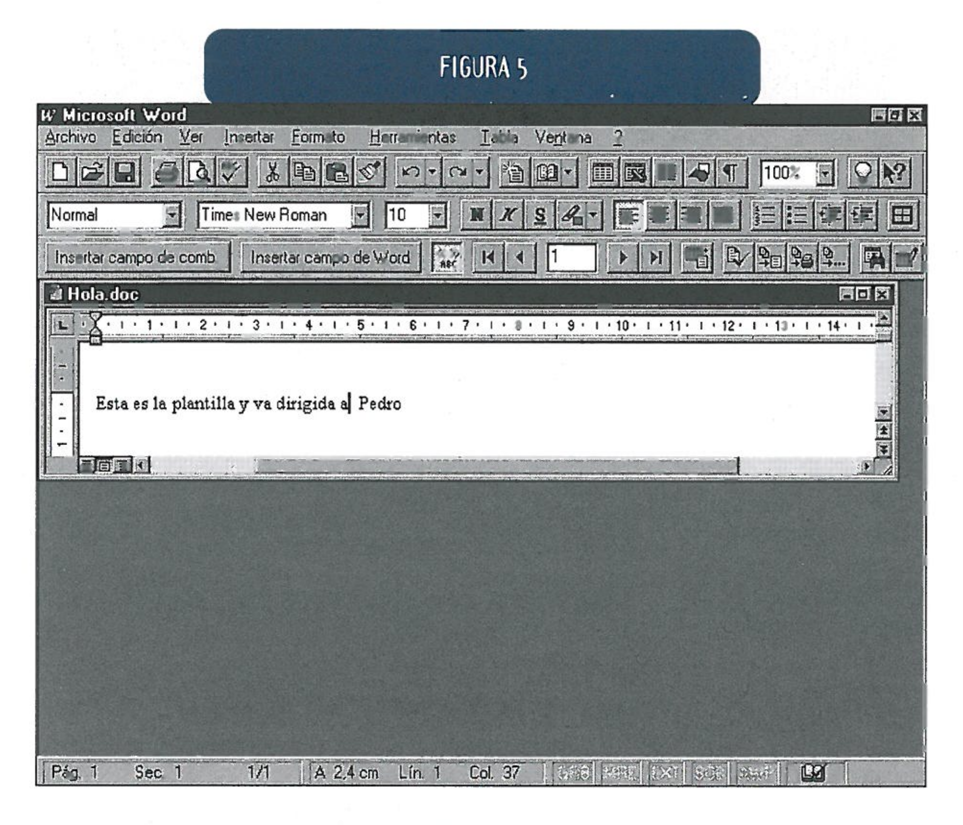
```
NombreObjeto.ArchivoAbrir nombre_documento
```

En cambio, en el programa los parámetros no tienen palabra clave sino que son posicionales:

```
ArchivoAbrir .Nombre = nombre_documento
```

Afortunadamente, la posición de los argumentos en el *Help* en línea de sintaxis para la mayoría de las entradas de instrucciones y funciones descritas en el *Help* de

FIGURA 5



WordBasic, es indicada correctamente. No obstante, hay algunas excepciones. Para obtener más información será necesario consultar el *Kit* del Programador de Microsoft Word, o cualquier otra referencia. Ello es debido a que estamos tratando con dos tipos de lenguajes Basic: Visual Basic y Word.Basic, siendo Word el único componente de Microsoft Office 95 que no incorpora el motor Visual Basic para Aplicaciones (VBA).

El objeto para trabajar con OLE Automatización viene con la clase *Word.Basic* y crearemos un objeto de esta clase. Es importante distinguir el objeto de OLE Automatización *Word.Basic*, de los objetos documentos de Word (Clase *Word.Document*). En el programa se va a tratar con el objeto *Word.Basic* y a través de éste se realizarán las acciones deseadas sobre objetos *Word.Document*, con lo cual aunque los documentos existan en el programa, se creará un objeto de la clase *Word.Basic*. La creación de objeto se hará con la función *CreateObject*:

```
Dim objFacturaWord As Object
Set objFacturaWord = CreateObject("Word.Basic")
```

Para trabajar con OLE Automatización, no solo es necesario conocer los métodos y propiedades que nos suministra cada servidor, sino que además, es necesario tener en cuenta las diferencias de comportamiento de cada Servidor

Podemos obtener información acerca del documento con la función CombinarFuente de Datos que en función del valor que se le pase como parámetro nos da la información que aparece en la Tabla 1. La combinación de documentos se realiza con la función CombinarCorrespondencia, esta función admite un juego de parámetros con los que podemos gobernar el tipo de correspondencia. En el programa se ha indicado que la fusión se realice sobre un documento nuevo, 2 parámetro puesto a 0, y que se impriman todos los registros, tercer parámetro con valor 0.

Otras opciones de la combinación son: permite dirigir la salida a impresora, fax, etc, seleccionar un rango de los registros del documento fuente se puede incluso extraer datos de Access o de Excel y realizar con ellos la fusión del documento.

Algunas diferencias

Para trabajar con OLE Automatización, no solo es necesario conocer los métodos y propiedades que nos suministra cada servidor, sino que, además, es necesario tener en cuenta las diferencias de comportamiento de cada Servidor, y estos comportamientos no son iguales para cada versión del mismo servidor, aunque en el caso de Microsoft Office estas diferencias tienden a ser menores en cada versión del producto. Veamos alguna de ellas que existen en los dos servidores con que hemos realizado el ejemplo.

- Word no crea una nueva instancia si ya existe una previamente arrancada, utilizando la instancia arrancada.

- Excel por el contrario crea una nueva instancia.

Ambos comportamientos tiene sus pros y sus contras. En el caso de utilizar la instancia arrancada se consumen menos recursos, pero puede crear confusión en los usuarios de nuestro programa si éstos están trabajando simultáneamente con el procesador de texto. Si se utiliza una ins-

tanciación distinta para cada servidor, nuestros programas consumirán más recursos. En cualquier caso, será necesario definir en tiempo de diseño cuál será el comportamiento de nuestro programa e investigar cuáles son las condiciones previas a la ejecución de los mismos; investigar si existen instancias en ejecución del servidor y proceder en consecuencia para obtener siempre un resultado homogéneo.

Otras condiciones a tener en cuenta al iniciar nuestro servidor son:

- Ver si se inicia con un documento nuevo o es necesario abrirlo.
- Ver si el servidor OLE se hace visible al usuario.

También es necesario evitar que el servidor se comunique con el usuario; por ejemplo, si nuestro programa modifica un documento, es necesario evitar que al cerrar el servidor (o el documento), éste nos pregunte si deseamos salvar el documento, lo que ocurrirá si modificamos nuestra hoja con las siguientes sentencias:

```
Set celdaWork = hSheetFacturacion.Cells(6, 1)
'celdaWork.Value = ""
'Hoja.Application.Quit
```

Si antes no salvamos el contenido con al método save:

```
Set celdaWork = hSheetFacturacion.Cells(6, 1)
'celdaWork.Value = "Cliente F"
'Hoja.Application.Save
'Hoja.Application.Quit
```

Logrando así evitar que Excel se dirija al usuario de forma externa al programa. Word no pregunta si se quieren guardar los cambios, perdiéndose éstos si no se guardan antes, al cerrar la aplicación o el documento. Si quisiéramos que Excel sea visible al usuario, tendremos que poner la propiedad Visible de la hoja a True. Todo esto sólo es una muestra de la diferencia de comportamiento entre dos aplicaciones servidoras OLE que pertenecen a un mismo grupo de programas, por lo que hay que esperar mayores diferencias de comportamiento si las aplicaciones servidoras han sido realizadas por distintas empresas o no forman parte de un paquete integrado.

Lograr un comportamiento definido de los servidores requiere realizar muchas pruebas hasta lograr el resultado definitivo. Esta parte del desarrollo de este tipo de aplicaciones es la parte menos agradable del desarrollo de programas con OLE Automatización.

Tabla 2. Errores Interceptables de Visual Basic para OLE.

429	El servidor de Automatización OLE no puede crear el objeto
430	Esta clase no acepta Automatización OLE
432	No se ha encontrado el nombre de archivo o el nombre de clase durante la operación de automatización OLE
438	El objeto no admite esta propiedad o método
440	Error de Automatización OLE
443	El objeto de automatización OLE no tiene un valor predeterminado
445	El objeto no admite esta acción
446	El objeto no admite argumentos con nombre
447	El objeto no acepta la configuración local actual
448	Argumento con nombre no encontrado
449	El argumento no es opcional
450	Número incorrecto de argumentos
451	El objeto no es una colección

Gestión de errores

Un control sobre un objeto no es completo si no tratamos la información acerca de los errores que nos suministre.

El tratamiento de errores se realiza con la sentencia *On Error* y los valores del objeto *Error*:

```
On Error GoTo TrataError
Set objFacturaWord = CreateObject("Word.Basic")
```

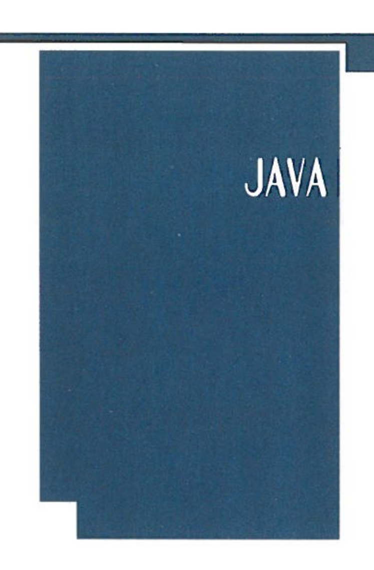
```
TrataError:
	Select Case Err.Number
		Case 440
		Case 443
		Case 445
			.
			.
		Case Else
	End Select
	Exit Sub
End Sub
```

La tabla 2 contiene los errores de Visual Basic para Automatización OLE.

Conclusiones

En nuestra curva de aprendizaje de conocimientos OLE hemos dado un paso más en el control de los servidores OLE, pero este aumento de control no nos ha sido gratuito, pues hemos debido de explorar en el interior de cada servidor para conocer sus métodos y propiedades. En el próximo artículo crearemos nuestro propio servidor OLE, que es una de las grandes diferencias que la versión 4 de Visual Basic tiene con respecto a las versiones anteriores.

Constructores, composición, herencia, ... y algo más.



Oscar Prieto Blanco

La Fortuna siempre ha sido una esquiva compañera del programador. En lenguajes anteriores, como el C (de sintaxis bastante libre), la existencia de variables sin inicializar en el devenir del código de un programa ha producido innumerables *bugs*, cefaleas, divorcios, crímenes, etc.

Hoy tomaremos del arsenal provisto por Java, las armas necesarias para evitar tanta tragedia. Estudiaremos el concepto de constructor, el cual nos brinda la oportunidad de dar valores iniciales a los atributos de nuestros objetos. Permitiéndonos, de paso, realizar algún tipo de tarea que necesitemos ejecutar antes de que el compilador nos proporcione acceso al objeto creado.

Pero debemos conocer armas más poderosas, si queremos preservar nuestra cordura, así que entraremos en contacto con los mecanismos de la composición y la herencia.

Estas técnicas nos permitirán reutilizar código de una manera más efectiva que el clásico cortar y pegar, tan arraigado en la programación procedimental y que a la larga producía un código pastiche, farragoso y difícil de mantener.

Pero antes de continuar tendremos que enfundarnos el mono, y como buenos "pinches" nos dedicaremos a la construcción.

Constructores

El constructor es un tipo especial de método, que siempre es llamado cuando creamos una instancia de una clase. Como cualquier método normal puede tener argumentos, pero no admite la especificación de un valor de retorno (ojo, incluso aunque pongamos *void)*. Para facilitar las cosas al compilador se decidió que su nombre siempre tendría que ser igual al de la clase a la que inicializa (como ocurre en C++). Al igual que una función normal, el constructor admite la sobrecarga, es decir, gozamos de la posibilidad de poder crear un objeto de múltiples maneras. Como ejemplo de lo visto hasta ahora, veamos la definición de una clase que simule a un contador:

```
class Counter {
// Establecemos el rango de valores.
static private final int MAX =
  (int)System.currentTimeMillis()%99L;
static private final int MIN = 0;
// Valores máximo, mínimo y actual
private int maxCount, minCount, numCount;
/////// El constructor principal ///////
public Counter(int minCount, int maxCount){
  this.maxCount = maxCount;
  this.minCount = minCount;
  numCount     = minCount;
}
////// Constructor alternativo //////
public Counter(int maxCount)
{ this(MIN, maxCount); }
////// El constructor por defecto //////
```

¡Ser o no ser: he aquí el problema! ¿Qué es más levantado para el espíritu: sufrir los golpes y dardos de la insultante Fortuna, o tomar las armas contra un piélago de calamidades y, haciéndoles frente, acabar con ellas?
SHAKESPEARE.
Hamlet.

Para definir una constante debemos utilizar la palabra reservada final y seguidamente asignarle un valor

```
public Counter()
{  this(MIN, MAX);        }
////// Información sobre el estado //////
public boolean isAtMax()
{  return numCount == maxCount;  }
public boolean isAtMin()
{  return numCount == minCount;  }
////// Operaciones del contador  //////
public void increment()
{  if (!isAtMax()) numCount++;   }
public Counter inc()
{  increment(); return this;   }
public void decrement()
{  if (!isAtMin()) numCount—;   }
public void reset()
{  numCount = minCount; }
public int countIs()
{  return numCount;    }
} // Fin de la clase Counter
```

Este ejemplo, además de ilustrar el uso de la sobrecarga de constructores, nos enseña los distintos usos de la palabra reservada *this*. Esta *keyword* no es más que un *handle* al propio objeto que la está utilizando (concepto similar al puntero *this* en C++). Una construcción muy típica con *this* se da en las sentencias *return*: puesto que contiene una referencia al propio objeto, podemos encadenar varias llamadas seguidas, como se aprecia en el siguiente fragmento:

```
// Utilizo el const. por defecto.
Counter cnt = new Counter();
// Incremento en tres el contador.
cnt.inc().inc().inc();
```

Cuando escribimos varios constructores para una clase, a veces ocurre que parte de un constructor incluye el código completo de otro. Para evitar una duplicación de código, la *keyword this* nos ayuda en esta tarea, como se ve en el ejemplo anterior: Counter(){ this(MIN, MAX);} Hay que precisar que esta técnica sólo puede ser aplicada desde un constructor. El tercer uso de *this* es mostrado en el constructor principal. Debido a que los argumentos pasados a él, tienen el mismo nombre que los atributos de la clase, una manera cómoda de diferenciarlos es aprovechar la oportunidad que nos brinda *this* de acceder al propio objeto:

```
this.maxCount = maxCount;
this.minCount = minCount;
```

¿Qué ocurre si en una clase no especificamos ningún constructor? Sencillamente, el compilador sintetizará uno sin argumentos para nosotros (constructor por defecto). Este constructor no afectará a los atributos de nuestra clase, tan solo realizará una llamada al constructor por defecto de la clase base (veremos más sobre esto cuando estudiemos la herencia). Un punto a tener en cuenta es que si definimos un constructor con argumentos, el compilador ya no nos creará automáticamente uno por defecto sin ellos; deberemos codificarlo manualmente.

En el ejemplo de esta sección podemos observar que Java nos proporciona varias alternativas al uso de constructores. Por ejemplo, tenemos la garantía de que cualquier variable será apropiadamente inicializada. En caso que sea una variable local:

Tabla 1. Valores por defecto.

Tipo	Valor
boolean	false
char	'u000' (null)
byte	(byte)0
short	(short)0
int	0
long	0L
float	0.0f
double	0.0d
handle	null

```
void func(){ int j; j++; }
```

se generará un error en tiempo de compilación si se nos olvida asignar algún valor inicial. En cambio, si es un dato miembro de una clase, tenemos asegurado, que si no es inicializado, el compilador nos proporcionará un valor por defecto (ver tabla 1).

En las primeras líneas de la clase Counter se observan un par de detalles interesantes:

```
static private final int MAX =
        (int)System.currentTimeMillis()%99L;
```

Para definir una constante debemos utilizar la palabra reservada *final* y seguidamente asignarle un valor. No hay que olvidar que, en Java, no podemos crear constantes como variables locales, deberán ser, obligatoriamente, miembros de clase. En este caso, hemos utilizado el especificador de acceso *private* que nos permite el más alto grado de protección. Los campos privados son accesibles a todos los métodos dentro de la clase. Sin embargo, impiden el acceso a cualquier otra clase, aunque sean derivadas suyas. Si usamos el especificador *private protected,* las clases derivadas sí podrán acceder a esos campos, pero ninguna clase más tendrá ese privilegio.

El mecanismo más simple para poder reusar el código de una clase consiste en introducir un objeto de dicha clase dentro de otra

Es posible, incluso, inicializar una variable miembro mediante el valor retornado por una función. Más todavía, esta función puede tener parámetros con la única condición que sean inicializados previamente a la llamada de dicho método.

Como finalización a este recorrido por el mundo de los constructores, vamos a echar un vistazo al orden de inicialización de los miembros de una clase. Este orden seguirá estrictamente el mismo que hayamos tomado al definir dichas variables dentro de la clase. Incluso si las definiciones de estas variables están dispersas entre las definiciones de los métodos. Estos datos miembro son inicializados antes de que sea llamado cualquier método, aunque sea el constructor. ¿Y qué ocurre con los datos estáticos?. ¿Cuándo es reservada localización en memoria para ellos? Las variables estáticas se comportan igual que las normales, con la diferencia que tienen mayor prioridad, lo que significa que, en la siguiente clase:

```
class nothing{
int first = 1;
static int second = 2;
}
```

aunque según el orden de definición, debería inicializarse *first* primero, ya que *second* es estática, el compilador localizará espacio en memoria para ella en primer lugar. En cuanto a la segunda pregunta, como los datos estáticos se comparten entre todas las instancias de una clase, la reserva de memoria se produce la primera vez que un objeto es declarado.

Es más, aunque no creemos ningún objeto, utilizando solamente funciones estáticas de una clase, en la primera llamada hecha a uno de dichos métodos los datos estáticos de la clase serán inicializados (cosa, por otro lado, bastante lógica, puesto que esta función estática podría hacer uso de los miembros estáticos de la clase).

Como ayuda para clarificar estos últimos conceptos, el lector puede ejecutar el Listado 1 (en el CD-ROM se corresponde con learnInitialization.java), donde se juega con la creación e inicialización de diversos objetos estáticos y normales, así como, con su orden de colocación, imprimiéndose resultados, que nos ayudarán a clarificar las ideas anteriormente expuestas.

Composición

El mecanismo más simple para poder reusar el código de una clase consiste en introducir un objeto de dicha clase dentro de otra. Nuestra nueva clase podrá contener cualquier número y tipo de otros objetos, hasta que alcancemos la funcionalidad deseada.

Este concepto se conoce como composición y nos proporciona un alto grado de flexibilidad, ya que podemos esconder los detalles de la implementación (especificando, por ejemplo, *private* a nuestros objetos), del interfaz provisto al usuario de la clase. De esta forma, podemos cambiar la manera en que nuestra clase trabaja, sin tener que afectar a código que hubiera sido escrito con alguna versión anterior de dicha clase. En el Listado 2 (en el CD-ROM Kiosko.java) podemos ver un ejemplo de composición: la clase *newspaper* ha sido formada como un agregado de un objeto *String* y un objeto *Date*. Asimismo, nos muestra el uso de la clase contenedora *Stack*.

Sólo podemos definir constantes como miembros de clases

Este ejemplo introduce unos cuantos conceptos que no habían sido tratados hasta ahora en esta serie, así que realicemos las presentaciones:

- La clase base Object.

La función *toString()* tiene como misión permitir una representación tipo *String* de cualquier objeto que manejemos. Este método es llamado automáticamente cuando un objeto se encuentra dentro de una sentencia de impresión por el dispositivo estándar de salida. Un ejemplo:

```
Object o = new Object();
// Estas sentencias son equivalentes:
System.out.println("Object = "+ o);
System.out.println("Object = "
        + o.toString());
```

El método *toString()* se encuentra definido en la clase *Object*, pariente último de todas las demás en Java. Esto im-

Listado 1

```
// learnInitialization.java
// Aprendizaje del orden de
// inicialización,
// de objetos estáticos y no estáticos.
// Atajo para impresión por pantalla
class P {
  static void prt(String s)
  {   System.out.println(s); }
}
// Vamos a crear unos cuantos

class Obj {
  Obj(int number)
  {   P.prt("Obj(" + number + ")");    }
  void funcObj(int seed)
  {   P.prt("funcObj(" + seed + ")");   }
}
// Territorio de pruebas para
// incializaciones.
class Test2 {
  static Obj o1 = new Obj(1);
  Test2()
  {   P.prt("Test2()");  o2.funcObj(1); }
  void funcTest2(int seed)
  {   P.prt("funcTest2(" + seed + ")"); }
   static Obj o2 = new Obj(2);
}
// Otro territorio de pruebas.
class Test3 {
  Obj o3       = new Obj(3);
  static Obj o4 = new Obj(4);
  Test3()
  { P.prt("Test3()");    o4.funcObj(2); }
  void funcTest3(int seed)
  { P.prt("funcTest3(" + seed + ")");    }

  static Obj o5 = new Obj(5);
```

Listado 1. Continuación

```
}
// Otro territorio más.
// ¡ Esto es la guerra !
class Test4 {
  static Obj o6;
  static Obj o7;
  // Otra construcción para inicializar
  // miembros estáticos.
  static {
     o6 = new Obj(6);   o7 = new Obj(7); }
  Test4()
  {    P.prt("Test4");  }
  static void write(String s) {    P.prt(s);
      }
}
// Programa principal:
public class learnInitialization {
  public static void main(String args[]) {
    P.prt("Creación nuevo Test3() en
        main");
    new Test3();
    t2.funcTest2(1);
    t3.funcTest3(1);
    P.prt("Utilización de una func.
        estática");
    Test4.write("No existe nigun objeto
        Test4, pero");
    P.prt("sus miembros estáticos, han
        sido incializados");
    Test4.write("con la primera llamada
        a Test4.write()");
    P.prt("Ahora creamos un obj. Test4");
        new Test4();
    }
    static Test2 t2 = new Test2();
    static Test3 t3 = new Test3();
}
```

plica que cualquier función en ella definida, siempre que no sea privada o contenga el especificador de acceso *final* (que impide la redefinición de un método en clases derivadas), permanecerá disponible para cualquier clase que creemos, que la podrá adaptar a sus necesidades. Por curiosidad, veamos la definición de *toString()* en la clase *Object*:

```
public String toString(){
        return getClass().getName() + "@"+
          Integer.toHexString(hashCode()); }
```

Cada clase en Java tiene un descriptor de clase. La clase *Class* representa este descriptor, que puede ser accedido usando el método *getClass()* de la clase *Object*. Aunque este valor retornado no puede ser modificado, sí puede ser usado para obtener información.

La clase Object es la madre de todas en Java

Por ejemplo, el método *getName()* retorna el nombre de la clase.

Por otro lado, el método *hashCode()* de la clase *Object* devuelve un *hashcode*. ¿Y qué es eso? Pues..., a cada objeto en Java le corresponde un *hashcode*, que es un número que suele ser diferente para distintos objetos y que es utilizado cuando almacenamos objetos en tablas de dispersión (*hashtables*).

En pocas palabras, una tabla *Hash* extiende el concepto de índice numérico para acceder a un elemento, utilizado en los *arrays* o en la clase *Vector*. Hasta el punto de permitir ligar el objeto que tenemos que almacenar con otro, que suele ser conocido como llave (*key*).

Posteriormente podremos acceder al objeto almacenado mediante el objeto *key*. Un ejemplo típico sería la implementación de un diccionario, haciendo corresponder a cada palabra (*key*) su definición.

Introducción a las excepciones

El lugar ideal para localizar un error se sitúa en tiempo de compilación. Pero siempre existirán errores que se deslizan hasta el momento de la ejecución (*runtime*). En lenguajes anteriores, como el C, la única manera de detectar estos últimos era seguir algún tipo de formalismo, aceptado tácitamente por todos los programadores.

Típicamente, había que seguir la pista a valores devueltos por funciones o inspeccionar algún *flag*, para obtener información acerca de la naturaleza del error.

Estos enfoques demostraron varios problemas. Por un lado, los programadores no solían chequear los valores de retorno (los errores son problemas de otros, mi código es demasiado bueno, pensaban). Y por otro, el hecho de mirar el valor retornado para tomar medidas, cada vez que se usaba una función problemática, transformaba, en poco tiempo, el código escrito en una especie de *zombi* (vida propia, poco manejable y escasa inteligencia).

Java utiliza un manejador de excepciones para el tratamiento de los errores de ejecución

El mecanismo que usa Java es similar al empleado en C++, ADA o Delphi, basándose en una construcción conocida como *exception handler* (manejador de excepciones) o *try/catch block.* Estudiemos un ejemplillo básico:

```
float Div(int num, int den)
        throws ArithmeticException
{
        if(den == 0)
            throw new ArithmeticException();
        else return num / den;
        }
    void otraFuncion(int n, int d){
        // ...
        String s1 = "5";
        String s2 = "Debería ser un número";
try{
        System.out.println((float)Div(n, d));
        System.out.println("S1 es igual a: "+
                        Integer.parseInt(s1) );
        System.out.println("S2 es igual a: "+
                        Integer.parseInt(s2) );
}
catch(ArithmeticException e){
    System.out.println(
    "Error, división por cero");
}
catch(NumberFormatException e){
System.out.println
("Formato erróneo");
}finally{
System.out.println
("Este código siempre se ejecuta");
        }
    }
```

En la definición del método Div() aparece la expresión:

```
throws ArithmeticException
```

De esta forma especificamos, avisamos al usuario de este método, que dentro de él se puede producir un error tipo división por cero o cálculo del módulo de cero. Se suele decir que dicho método "lanza" o "arroja" una excepción.

En el cuerpo de la función nos encontramos la sentencia:

```
throw new ArithmeticException();
```

Si el denominador es 0, se ejecutará esta línea y ocurrirán un cierto número de sucesos. Primero se creará el objeto excepción en el *heap* mediante *new*. Entonces la ruta de ejecución seguida hasta este momento se detendrá, arrojando un *handle* del objeto excepción hacia un contexto de más alto nivel (el bloque *catch*). Entonces es cuando toma el mando el mecanismo de manejo de excepciones, que buscará un lugar donde seguir la ejecución del programa. Aquí es donde toma sentido el bloque:

```
try{
//Código generador de excepciones
}
```

Cualquier excepción arrojada dentro de este alcance será recogida y comparada con los argumentos de las distintas sentencias *catch* que la siguen:

```
// Sólo se ejecutan si es
// arrojada alguna excepción.
catch(ArithmeticException e){
  // código si exc. aritmética
}
catch(NumberFormatException e){
        // Código si exc. de formato
}
```

El manejo de errores es una asignatura pendiente en muchos lenguajes

La construcción *finally*{/*...*/} nos permite ejecutar la pieza de código entre llaves, tanto si una excepción es lanzada en el bloque *try*, como si no. En ocasiones este comportamiento puede ser de utilidad, por ejemplo, sin necesitamos cerrar un fichero, o una conexión de red.

Para acabar con esta pequeña introducción a las excepciones, hay que comentar que existe un grupo de ellas, derivadas de la clase *RuntimeException* (ver Tabla 2) que son automáticamente lanzadas por Java, por lo que no es necesario incluir manejadores para ellas.

Fijándonos en su significado podemos ver que representan ciertos errores típicos de programación, siendo de gran ayuda para depurar nuestro código. Veamos qué pasa si no las manejamos:

```
// prueba.java
public class prueba{
static void func(){
        throw
    new RuntimeException("Desde func");
  }
  static void otrafunc(){ func(); }
  public static void main(String args[])
```

Listado 2

```
// Kiosko.java -"  Comprendiendo la composición y, de paso,
//             una introducción a la entrada/salida y al
//             manejo de excepciones.
import java.util.Stack;
import java.util.Date;
import java.util.EmptyStackException;
import java.io.IOException;

class Newspaper {
  Date date;
  String publisher;
  Newspaper(String pub, Date d)
  {  date = d;  publisher = pub; }
  Newspaper(String pub)
  {  this(pub, new Date());  } // usamos el día actual.
  // toString() es una func. de la clase base Object.
  public String toString()
  {  return (publisher + ", " + date); }
}

class P{  // El clásico atajo de escritura.
  public static void prt(String s)
  {  System.out.println(s);  }
}

public class Kiosko {
  public static void main(String[] args) throws IOException
  {
    Stack newspapers = new Stack();
    // Añadimos unos elementos:
    Newspaper LEX = new Newspaper("L'Express");
    Newspaper LMD = new Newspaper("Le Monde");
    Newspaper LRP = new Newspaper("La Republica");
    // Los guardamos en el Stack
    newspapers.push(LEX);
    newspapers.push(LMD);
    newspapers.push(LRP);
```

```
      { otrafunc();      }
}
```

Cuando usamos un stream de salida, nosotros somos la fuente de los bytes

Como salida de este programa obtenemos un listado de todas las funciones por donde ha pasado la excepción antes de ser manejada:

```
java.lang.RuntimeException: Desde func
      at prueba.func (prueba.java:4)
            at prueba.otrafunc (prueba.java:6)
         at prueba.main (prueba.java:8)
```

Sobre Entrada/ salida(IO).

Se denomina *stream* a una corriente, a una secuencia de *bytes,* que puede ser leída o escrita durante el transcurso del tiempo y que fluye entre un origen y un destino. Más sencillamente, la fuente y el destino de un *stream* pueden verse como productores y consumidores de *bytes*. La librería de clases de Java para el manejo de *streams* se encuentra divida en entrada y salida. Gracias a la herencia, todas las clases derivadas de *InputStream* contienen métodos básicos, tipo *read(),* mediante los cuales un consumidor lee un *byte* o una secuencia de *bytes* procedente de alguna fuente.

Es importante entender que la identidad de la fuente productora de bytes (podría ser un fichero, una cadena de caracteres, un *array* de *bytes,* etc.) y la manera de circulación y transporte de los bytes son irrelevantes. Cuando usamos un *stream* de entrada, nosotros somos el destino de esos *bytes.* De igual manera, todas las clases derivadas de *OutputStream* contienen métodos, tipo

Listado 2. Continuación

```
// Búsqueda de elementos
int where = newspapers.search(LEX);
P.prt(LEX + " necesita " + where + " pop(s)");
// Nuestro primer encuentro con las excepciones:
try
{ // Obtención del elm. en la cima, pero sin extracción
        Newspaper top = (Newspaper) newspapers.peek();
        P.prt("Top contiene: " + top);
        boolean f = false;
        do
  { top = (Newspaper) newspapers.pop();
          P.prt("Popped off: " + top);
          P.prt("¿Otro pop? SI-"pulsar y, NO-"cualquier tecla");
          // Lectura desde el dispositivo standard de entrada
          switch(System.in.read()){
                  case 'y':  f = true;  break;
                  default:   f = false;
          }
          System.in.skip(1); // Nos saltamos el retorno de carro
        }while (f);
     }catch(EmptyStackException e)
        { P.prt("Cuidadín, no hay elementos en el Stack"); }
  }
}
```

write(), gracias a los cuales una fuente (productor) escribe un *stream* de *bytes* en algún destino. De nuevo, la identidad del destino y la manera de transportar y almacenar los *bytes* son irrelevantes.

Cuando usamos un *stream* de salida, nosotros somos la fuente de los *bytes*. Dado el enorme montón de clases con que cuenta Java para el manejo de IO, las dedicaremos un próximo artículo. Por hoy nos basta con conocer que mediante la variable estática *System.in* (de tipo *InputStream)* tenemos acceso a los bytes originados en el dispositivo estándar de entrada, normalmente el teclado. Ahora ya deberíamos entender las siguientes líneas del Listado 2:

```
switch(System.in.read()){
      case 'y':        f = true;  break;
      default:         f = false;
}
System.in.skip(1);
```

Cuando usamos un stream de entrada, nosotros somos el destino de esos bytes

System.in implementa un *stream* de entrada con un *buffer* de 2048 *bytes*. Mediante la función *read()* obtenemos un *byte* desde el *stream* de entrada que es devuelto por la función como un *int*. Puesto que al entrar cualquier carácter por el teclado pulsamos la tecla *intro*, su valor también se guarda en el *buffer*. Esta es la razón por la que debemos saltarnos dicho carácter mediante la función *skip()*.

Para despedirnos de los *streams*, veamos un típico programilla que nos sirve para contar el número de palabras, caracteres y líneas, tanto de un texto introducido por el teclado (Ctrl-Z para finalizar su entrada), como de un fichero o conjunto de ficheros, especificados en la línea de comandos.

```
// WordCount.java:
import java.io.*;
public class WordCount{
        public static int words = 0;
        public static int lines = 0;
        public static int chars = 0;
              public static void wc(InputStream f)
throws IOException
{
int c = 0;
boolean lastNotWhite = false;
String whiteSpace = " \t\n\r";
while((c = f.read()) != -1) {
        chars++;
        if(c == '\n') lines++;
        if(whiteSpace.indexOf(c) != -1){
              if(lastNotWhite){
                          words++;
                          lastNotWhite = false;
              }
        }else{ lastNotWhite = true;}
      }
    }
public static void main(String args[])
throws IOException{
if(args.length == 0) wc(System.in);
else   // ficheros a contar
        for(int i=0; i< args.length; i++)
wc(new FileInputStream(args[i]));
          System.out.println();
          System.out.println(lines
    + " " + words + " " + chars);
        }
} // Fin de la clase WordCount
```

En este código ocurre una cosa curiosa: la función *wc()*, que especifica co-

mo argumento un objeto de tipo *InputStream,* acepta por un lado la variable *System.in* y por otro una instancia de la clase *FileInputStream* (la cual deriva de *InputStream* y nos permite utilizar como fuente de bytes a un fichero). ¿Por qué ocurre eso? Sencillamente, hay que tener en cuenta que uno de los aspectos más importantes de la herencia es que nos permite asignar un *handle* de una clase derivada a un *handle* de una de sus clases base. De esta manera podemos escribir una función como *wc(InputStream f)* y pasar como argumentos clases derivadas de *InputStream* que, por si fuera poco, es una clase abstracta (contiene el modificador de clase *abstract* o existe en su definición algún método solamente sin cuerpo).

Brevemente, una clase abstracta no puede ser instanciada (aunque sí puede contener *handles* de clases derivadas), el propósito de su existencia es proveer una interfaz, un esquema, que podrá ser implementado y refinado, por clases que deriven de ella.

Una clase abstracta no puede ser instanciada (aunque sí puede contener *handles* de clases derivadas)

Dado que en una clase derivada podemos redefinir (también conocido como sobrepasar u *override*) las funciones de sus clases base, ¿qué ocurrirá si asignamos un *handle* de esa clase derivada a uno de su clase base y ejecutamos un método sobrepasado?, ¿cuál será llamado, el de la clase derivada o el de la clase base? Ya hemos visto en el anterior ejemplo que será llamado el de la clase derivada. Esta propiedad se conoce como polimorfismo y Java lo soporta por defecto (no como C++, donde debíamos utilizar la palabra *virtual*).

Después de tanto hablar de clases base y derivadas creo que es un buen momento para tratar la herencia (ver artículo sobre este tema).

Herencia

Mediante la herencia creamos una nueva clase tomando como base una ya existente. La clase derivada hereda las propiedades de la clase base, incluyendo sus atributos (miembros de datos) y sus métodos, pero la clase base no es modificada. Imaginemos que queremos crear unas clases para manejar la tabla periódica. Como clase base se podría definir un objeto general llamado *Element*, que fuera

Mediante la herencia creamos una nueva clase tomando como base una ya existente

contenedor de todas las propiedades comunes que comparten los elementos de la tabla periódica:

```
public class Element extends Object {
  // Atributos
  public int atNumber;     //Número atómico
  public int massNumber;   //Masa atómica
  public double atRadius;  //Radio atómico
  public double density;   //Densidad
  public double specHeat;  //Q específico
```

Tabla 2. Principales excepciones derivadas de RuntimeException.

Nombre	Aparición
ArithmeticException	Error aritmético: división por cero, etc.
ArrayStoreException	Cuando intentamos almacenar objs. de un tipo, en un array que contiene objs. de algún otro tipo incompatible.
ClassCastException	Si intentamos realizar un casting hacia un tipo incorrecto.
EmptyStackException	Un objeto Stack está vacío y realizamos un pop.
NumberFormatException	Un método llamado ha recibido un formato de número ilegal.
ArrayIndexOutOfBoundsException	Usando un array, hemos seleccionado un elemento con un índice menor que cero o mayor o igual al tamaño del array.
NegativeArraySizeException	Se intenta crear un array de tamaño negativo.
StringIndexOutOfBoundsException	En una String intentamos seleccionar un índice menor que cero, o mayor o igual que el tamaño de la cadena.
NullPointerException	Intentamos seleccionar un miembro de un objeto usando un handle que no ha sido inicializado.

```
  // podríamos agregar más propiedades
  public Element() { }   //constructor
  // Función sobrepasada de Object
  public String toString()
  {  return new String("Element"); }
  // Comparación de dos elementos
  public boolean equals(Object e) {
    return
     (e!=null)&&(e instanceof Element) &&
     (atNumber==((Element)e).atNumber) &&
     (massNumber==((Element)e).massNumber);
  }
}
```

El código de cada clase se encuentra en un fichero separado que no es cargado hasta que la clase no es construida en el programa

A partir de esta clase se podría derivar tranquilamente cada elemento químico, dando un valor adecuado a cada uno de sus atributos. Por ejemplo:

```
public class Iron extends Element{
  atNumber   = 26;
  massNumber = 56;
  atRadius   = 1.72;
  // ... etc
}
```

Pero, recordando que en la tabla periódica los elementos se encuentran almacenados en grupos: metales alcalinos (grupo IA), alcalinotérreos (grupo IIA), gases nobles, etc., sería más útil escribir una clase para cada uno de dichos grupos y a partir de ellos, derivar los elementos que contiene. Un ejemplo:

```
public class AlkaliMetal extends Element{
  public AlkaliMetal() {}
  public String toString()
  { return new String("Alkali Metal"); }
}
```

Listado 3

```
// Lucifer.java:  Aprendiendo el orden de inicialización, cuando
//                mezclamos la herencia, los constructores y los
//                miembros estáticos.

class P{  // El clásico atajo
  public static int prt(String s)
  { System.out.println(s);  return 1;  }
}

class BadAngel{
  BadAngel()
  { P.prt("Constructor de un ángel malo."); }
}

class Devil extends BadAngel{
  private static int numDevil =
     P.prt("El número del Diablo: " + numOfBeast());
  protected int numTails =
     P.prt( "Primera inicialización de las colas" +
            " del Diablo, tiene sólo una cola.");
  static int numOfBeast()
  {  return (int)(Math.random()*666);}

  Devil(int k){
     numTails = k;
     P.prt("Dentro del constructor del Diablo.");
     P.prt("El Diablo tiene "+ numTails + " colas ahora.");
     }
}// Fin de la clase Devil

class Lucifer extends Devil{
  int numHorn =
     P.prt("Lucifer tiene un cuerno.");
  static int DNI =
     P.prt("El DNI de Lucifer: " + numOfBeast());
  Lucifer(){
     // Lo primero siempre, llamada al constructor
```

Listado 3. Continuación.

```
        // de la clase base, si no, error de compilación.
        super(numOfBeast());
        P.prt("Dentro del constructor de Lucifer.");
        P.prt("Lucifer tiene "+ numTails + " colas.");
    }
    public static void main(String args[])
    { Lucifer l = new Lucifer(); }
}// Fin de la clase Lucifer
```

```
// Haríamos lo mismo con el resto de los grupos:
// public class Halogen extends Element{...}
// public class NobleGas extends Element{...}
// etc...
// Y ahora, por fin, los elementos:

public final class Potassium
        extends AlkaliMetal {
  // Estos valores ya no son modificables:
  public final static int atNumber   = 19;
  public final static int massNumber  = 39;
  public final static double atRadius = 2.77;
  public final static double density  = 0.862;
  public final static double specHeat = 0.75;
  // Constructor
  public Potassium() {
   // Actualizo los valores de la clase base:
   super.atNumber   = atNumber;
   super.massNumber = massNumber;
   super.atRadius   = atRadius;
   super.density    = density;
   super.specHeat   = specHeat;
  }
  // Retorna el símbolo del elemento
  public String toString()
  { return new String("K")}
}
```

En la anterior definición de clase, aparece la palabra *final* como modificador de clase. Gracias a ella, avisamos al compilador que no permitimos a nadie construir una clase derivada de *Potassium* (debemos meditar siempre mucho el hecho de no permitir la herencia, en este caso, no podríamos, por ejemplo, trabajar con isótopos de los elementos, ya que no tendríamos posibilidad de especializar más la clase, añadiendo quizás, un *array* de *floats* para guardar sus distintas masas atómicas). También aparece la *keyword super,* de funcionamiento similar a *this.* Por mediación de *super* podemos acceder a miembros (datos o funciones) de las clases base, cuando la sintaxis es ambigua.

Gracias a la palabra final avisamos al compilador que no permitimos a nadie construir una clase derivada a partir de otra clase

Ahora es el momento de echar una miradita al Listado 3 (en el CD-ROM Lucifer.java). Este programa nos ilustra el proceso de inicialización de clases cuando mezclamos la herencia, los constructores y las variables estáticas. Veamos si podemos entender la salida de dicho programa: en muchos lenguajes tradicionales, el programa entero es cargado en memoria como parte de su proceso de *startup,* después se produce alguna inicialización de las variables globales y estáticas, y por último el programa comienza su ejecución. *Java* implementa otro procedimiento para *realizar la carga de un programa.* Normalmente, el código de cada clase se encuentra en un fichero separado. Este fichero no es cargado hasta que la clase no es construida en el programa, o más rigurosamente (teniendo en cuenta que pueden existir miembros estáticos), hasta el punto de su primer uso.

Así que la primera acción que ocurre, cuando ejecutamos Java sobre Lucifer, es que el cargador busca esta clase y comienza a cargarla. Al realizar esta tarea se da cuenta que Lucifer tiene una clase base, luego también la carga; este proceso seguiría hasta llegar a la clase *Object.* Seguidamente, se producen las inicializaciones estáticas en la clase raíz y después en sus derivadas (en este caso, Devil y después Lucifer). Y por último, empezarán a ejecutarse las inicializaciones de los datos miembro no estáticos de las clases bases, seguidos de sus correspondientes constructores, hasta que haya sido atravesada toda la jerarquía en orden descendente.

Conclusión

En este artículo hemos aprendido a inicializar correctamente nuestras clases mediante las distintas posibilidades que nos brinda Java. Después, con la excusa de estudiar la composición como método de reutilización de código, hemos navegado por los mundos de la clase base *Object*, el manejo de excepciones y la entrada/salida de un programa. Y por último, el uso de los *streams* nos ha servido para introducir el concepto de polimorfismo, como preludio al tratamiento del mecanismo de la herencia.

Contactar con el autor

Para cualquier duda, comentario, sugerencia o crítica se anima al lector a ponerse en contacto:

E-Mail myoprieto@redestb.es

LENGUAJE C

Programación avanzada en C

Jorge Figueroa

Actualmente muchos de los programadores profesionales usan para desarrollar sus aplicaciones el lenguaje C/C++. En este artículo se van a explicar aquellos aspectos de la programación de dicho lenguaje que siempre ocasionan problemas: las directivas del compilador, cómo se enlaza con ASM, normas del programador...

El C, como muchos sabemos, es un lenguaje que se caracteriza por ser muy flexible a la hora de definir y usar punteros, arrays y funciones. Como consecuencia de ello, entre otras cosas, se ha convertido en uno de los más usados en todo el mundo. En contrapartida nos encontramos con que se trata de un lenguaje donde la dificultad de interpretación de la sintaxis también aumenta según se eleva la complejidad de los programas que realizamos con él, pudiendo llegar a confundir al propio autor de los fuentes. También hay muchos aspectos de este lenguaje a los que normalmente no se hace mucho caso, consiguiendo que no saquemos todo el partido a las directivas y funciones de las que disponemos, así como a la capacidad de poder enlazar nuestros fuentes C con procedimientos propios de ensamblador.

Cuando una función tiene que devolver más de un valor puede hacerse bien usando variables globales o bien usando punteros

Todo lo mencionado se va a detallar en el presente artículo con el fin de que los programadores ya bastante iniciados en este lenguaje acaben de comprender puntos oscuros de su sintaxis, medios que nos ofrece y demás. Además, como complemento, se van a mencionar también las normas más básicas que debe tener presente todo programador que tenga intención de desarrollar algún programa extenso y complejo, puesto que una mala organización inicial en la creación del mismo puede dar al traste con todo el trabajo cuando lo tengamos muy avanzado a la hora de depurarlo o de pasarlo a otras plataformas que no sean la nativa sobre la cual se diseñó.

Punteros, definición y utilidad

Una de la palabras que más se usan en el lenguaje C (entre otros) es *puntero*. Los punteros son una parte imprescindible del programa y, a su vez, son la causa de los mayores problemas y errores que se producen en los programas que desarrollamos.

Un puntero es simplemente un número que corresponde a la dirección de memoria que apunta a una variable, búfer de memoria o array de datos y que se usa para poder hacer referencia a esa dirección.

En principio, podemos decir que los punteros en C se usan para los siguientes fines:

1) Cuando una función tiene que devolver forzosamente más de un valor (parámetro), sólo puede hacerse o bien usando variables globales, lo cual no es ni práctico ni profesional, o bien usando punteros, de forma que el valor que devuelve la función sea el puntero al búffer de datos donde se han almacenado los valores de retorno (eliminando así la limitación en cuanto a número y longitud de elementos retornados).

2) Gracias a los punteros podemos desplazarnos más fácilmente por el interior de arrays a la hora de manipular datos.

3) Con los punteros se hace posible la creación de tablas de datos indexadas, donde los elementos apuntan a su vez a otras tablas y búfers de memoria.

4) Gracias a ellos, podemos indicar punteros a arrays, variables y búfers de memoria como parámetros de las funciones que requieren sus direcciones así como también obtenerlos de las mismas. Ejemplo de ello es cuando damos el puntero de un gráfico a una rutina para que ésta pueda saber donde está almacenado el dibujo a copiar en memoria o también cuando obtenemos un puntero tras llamar a la función MALLOC para localizar un bloque (búfer) de memoria libre para poder utilizar.

Sintaxis básica y uso

Lo primero que debe tener memorizado el programador es cómo se definen los punteros y cómo se indican dentro del programa, puesto que ello es una de las principales causas de error sintáctica de compilación y de funcionamiento de los programas que realizamos.

> **Para definir un puntero, solamente tenemos que añadir un asterisco delante del propio nombre definido**

Así pues, sólo recordar las siguientes:

Para definir un puntero, solamente tenemos que añadir un asterisco delante del propio nombre definido. Ejemplos de ello son:

```
int *PUNT1;
char *DIR2;
char *pnt1,*pnt2;
```

Es muy importante tener presente siempre que el tipo con el que se definen no corresponde al número de bytes que forma el puntero, sino que todos son números de 32 bits. El tipo de dato que se le atribuye se usa únicamente para indicar al compilador a qué tipo de datos de memoria apuntará cuando lo usemos en nuestros programas. Así pues, si tenemos un búfer cuyo contenido son números de 4 bytes almacenados de forma secuencial (array de INTs), si queremos crear un programa que vaya uno por uno y sume a cada uno 10, tenemos que crear un puntero para apuntar a sus contenidos tipo **INT** (4 bytes). En el caso que hubiesen sido elementos de 1 byte cada uno, hubiésemos tenido que definir un tipo **CHAR**.

Otra ventaja que da el hecho de definir los tipos de datos que queremos usar con los punteros es que después, en el código de programa, no tendremos que preocuparnos por sumar el número de bytes que debemos aumentar para saltar de un elemento a otro, puesto que el compilador de forma automática ya realiza dicho cálculo. Así, por ejemplo, si realizamos un **PUNTERO++;** al cual hemos definido con el tipo **INT**, el propio compilador se encarga de sumar 4 al número contenido en puntero, o sea, que hace que Puntero apunte al siguiente elemento múltiple de 4 bytes.

Pese a todas las ventajas de poder definir el tipo de los punteros, queda

Tabla 1. Manipulación de array con punteros.

```
//--------------------
// TABLA 1:
// Ejemplo de limpieza de un array y
// un búfer
// usando punteros y del envío y
// recepción de
// parámetros tipo puntero a y desde
// funciones.
//----------------------
main()
 {
  int TABLA(1000; // Array 1000 elemen-
      tos
  int Conta=0;
  int *PUNT1;
  int *BUFFER1;
 // Limpia el Array con ceros.
  PUNT1= TABLA;
  for(Conta=0;Conta"1000;Conta++)
   *(PUNT1+Conta)=0;

 // Define y Rellena el Búfer con
 // "Conta"
  BUFFER1= malloc(40000);
  for(Conta=0;Conta"40000;Conta++)
   *BUFFER1=Conta;
  }
```

Al definir tablas de arrays como punteros, lo que estamos realizando es una tabla de números cada uno de los cuales corresponde a la dirección de inicio en memoria donde se ha guardado la cadena

también la posibilidad de que requiramos punteros que no sabemos a qué tipo de datos van a apuntar. Para ello, recordad que existe la definición de punteros tipo **void ***, que literalmente indica definición vacía, sin tipo, con lo que podremos usarlo con cualquier tipo de array que se nos presente.

En la Tabla 1 podemos ver el código C de un sencillo programa donde se manipula un array mediante punteros. Con lo que toda la sintaxis explicada queda clara.

Definición de arrays tipo puntero

Otra ventaja que permite el C con los punteros es que podemos definir tablas de cadenas de texto en forma de punteros de arrays con lo cual nos ahorramos muchísimo espacio de definición de datos, ya que, si no dispusiéramos de este sistema, sólo podríamos realizarlo mediante tablas bidimensionales ASCII donde cada línea horizontal debería poseer la longitud en elementos suficiente como para albergar la mayor cadena posible que pudiésemos necesitar almacenar dentro de ella, con lo cual quedarían muchos espacios vacíos, entre otras desventajas.

Al definir tablas de arrays como punteros, lo que estamos realizando realmente es una tabla de números cada uno de los cuales corresponde a la dirección de inicio en memoria donde se ha guardado la cadena (lo cual controla el compilador de forma automática). Así pues, cuando definimos **char *LISTA[3]= { "Arbol", "Casa", "Mesa"};**, lo que tenemos realmente es una tabla de 3 elementos llamada LISTA, cada uno de los cuales es un número de 4 bytes que corresponde a la dirección de memoria donde el propio programa ha almacenado cada una de las cadenas.

En la figura 1 podemos ver el dibujo gráfico comparativo de las dos formas de definición entre otros ejemplos de datos donde se puede observar las ventajas del uso del tipo puntero.

Los dobles punteros

Si profundizamos más en la capacidad de manejo de punteros del lenguaje C, descubriremos que es posible lo que podríamos llamar *anidamiento* de punteros, que consiste en realizar sumas de punteros relativos, permitiéndonos manejar tablas de datos de dos o más dimensiones por medio de punteros.

Un caso simple práctico del uso de punteros anidados sería la manipulación del contenido de una tabla ASCII bidimensional del tipo clásico char TABLA[40][50]; en una operación normal, haríamos referencia al cuarto elemento de la 3ª línea con **X=TABLA[3][4]**; pero si usamos correctamente la sintaxis de punteros, podemos crear la línea equivalente siguiente: **X= *(*(TABLA+3)+4)**;. ¿Como se entiende esto? Pues muy sencillo, si el asterisco significa <*el contenido de*>, tenemos que ***(TABLA+3)** significa <*el contenido del puntero al primer elemento de la tercera línea*>, puesto que como hemos dicho, el compilador se encarga de cambiar los operadores que se suman a un puntero por los valores correspondientes a su tipo, si hemos definido que cada línea de TABLA posee 50 elementos, el compilador lo convertirá de forma automática a *(TABLA+150).

Por otro lado, hemos sumado al puntero indicado en dicho paréntesis <*el puntero del paréntesis+ 4º elemento de dicha línea*>. Al ser cada elemento de 1 byte, el compilador sumará a lo anterior simplemente 4. Con todo esto tenemos que la línea escrita apunta al mismo elemento que la versión simple.

Aunque a simple vista parezca inútil esta forma de usar punteros, la verdad es que a la hora de programar, esta flexibilidad de sintaxis se hace muy apreciada.

Figura 1. Ejemplos de la memoria real ocupada por la definición de diversos tipos de datos.

Definición y uso de macros

Prácticamente todos los programadores de C conocen la existencia de la directiva #define, a la cual se considera una simple definidora de constantes (una especie de CONST). La realidad es que su utilidad va mucho más allá, puesto que nos proporciona la posibilidad de crear verdaderas macros con la capacidad de poseer incluso parámetros (argumentos) de igual forma que si se tratara de una verdadera función.

Así pues, tenemos que si definimos, por ejemplo, **#define ERROR printf("Generado Error\n");**, el simple hecho de escribir ERROR en una línea de código nos ahorra escribir la línea entera cada vez que necesitamos una salida de aviso por pantalla. Pero además, como hemos dicho, podemos incorporar argumentos, con lo que la definición siguiente: **#define ERROR(X) printf("Generado Error %d\n",X);** es totalmente correcta. Después, cuando dentro del programa, con tan sólo escribir ERROR(var1); podremos escribir un mensaje de error personalizado, ya que el compilador lo expandirá automáticamente a **printf("Generado error %d\n",var1);**.

Como se puede deducir, el uso de **#define** con argumentos se puede aplicar también a la generación de algoritmos matemáticos sin necesidad de tener que definir funciones con parámetros de entrada y salida. Así por ejemplo, para calcular el área de una esfera (4*PI*Radio^2), no hace falta crear una función para ello, o escribir cada vez todo el algoritmo. Tan sólo requerimos la siguiente línea: **#define AREA(Rad) (4*3.14*Rad*Rad)**. Posteriormente, cada vez que requiramos dicho cálculo, solamente tenemos introducir la línea **AREA(numero)**. Como se puede observar, #define es muy útil y el hecho de que acepte parámetros la hace muy flexible. El único problema que podemos llegar a encontrarnos es que existen algunos compiladores que sólo aceptan una línea de código para una definición, por lo que las macros quedarán bastante limitadas en este caso.

Tabla 2

```
//---------------------
//TABLA 2:
// Ejemplo de uso de directivas para
//compilación
// selectiva. En el ejemplo se puede
//compilar
// en modo DEPURADO o FINAL
//---------------------
Nºdefine DEPURADO
main()
{
 int Conta, N_B=0;
 Nºif defined(DEPURADO)
    printf("Empieza ejecuciónn");
 Nºendif

 for(Conta=0;Conta"1000;Conta++)
   N_B++;
 Nºif defined(DEPURADO)
    printf("Fin de la ejecución.n");
  Nºelse
    printf("Okn");
 Nºendif
}
```

Programas complejos

Cuando realizamos programas complejos, en ocasiones necesitamos crear programas que al convertirse en ejecutables deben sólo compilar algunas partes del código fuente dependiendo de algunas condiciones concretas.

Un ejemplo muy claro y real de su uso es poder realizar la compilación de un programa en modo depuración y en modo versión final. En esta situación, necesitaremos que el compilador sólo lea las líneas de monitorización de funcionamiento que hayamos incluido durante el desarrollo del programa (o sea, hasta la última versión beta) y que cuando se dé por acabado, se pueda hacer que no compile dichas líneas sin tener que recurrir a su eliminación de los fuentes *físicamente*.

Pues bien, para la situación explicada disponemos de tres directivas, que son **#if, #endif** y **#else**, junto con dos que están relacionadas indirectamente con ellas, que son **#define** y **#undef** y que ya conocemos (al menos una). Estas directivas, que muchos programadores tienen la costumbre de ignorar, son en realidad de una gran utilidad y la función de cada una de ellas puede ser deducida por su propia representación sintáctica y no se va a detallar. La única que queda un poco ambigua es **#undef**, que tan sólo sirve para eliminar una definición que tengamos hecha anteriormente y que esté en la memoria del compilador. El único dato importante que se debe tener en cuenta es que como parámetros usan definiciones que estén activas y de ahí se deriva que #define y #undef estén relacionadas con ellas.

Para resumir todo su funcionamiento, nada mejor que verlo en el ejemplo de la tabla 2.

El ejemplo queda muy claro. El código simplemente se encarga de incrementar mil veces un contador.

Las directivas **#if defined** detectan si la definición DEPURADO está presente (definida). En caso de existir, el compilador leerá también todo lo que incluyan en sus contenidos hasta llegar al **#endif**. El resultado práctico es sencillo de prever. Por otro lado, si dejamos la línea que define a DEPURADO, el programa se ejecutará con todos los mensajes que informan de su inicio y su finalizado. En caso de que eliminemos dicha definición (#undef), sólo se ejecutará el mensaje <OK>.



Declaraciones complicadas

Incluso una vez comprendido totalmente el uso y definición de punteros, arrays y relacionados, podemos encontrarnos fuentes C/C++ que realmente contienen líneas donde no sabemos exactamente lo que hay. Esto ocurre debido al hecho de estar presentes paréntesis, asteriscos y otros elementos que según cómo los combinamos, cambian el sentido de lo que representan, funcionalmente hablando.

A continuación ponemos algún ejemplo de definiciones complejas:

void **CADENA[];

Aunque parezca un error de sintáxis, la definición mencionada tiene sentido, y su explicación es que con ella definimos un array de punteros que apuntan a punteros definidos como tipo INT.

void *Func_1();

Esta línea significa que la Func_1 devuelve, tras ser ejecutada, un puntero tipo VOID (sin tipo).

void (*Func_1)();

En este caso, el simple hecho de añadir dos paréntesis a la definición cambia totalmente el sentido, pasando ahora a ser un puntero a una función que devuelve un puntero VOID.

int *(*Func_1)();

Este ejemplo, que hemos definido con un asterisco más fuera del paréntesis que en el ejemplo anterior, posee un contenido muy diferente. Ahora esta línea define un puntero a una función que devuelve un puntero del tipo INT.

void *(*CADENA[])();

Este ejemplo quizás es el más complejo de todos los explicados, aunque si lo analizamos detenidamente, veremos que sólo hace falta estudiarlo con cuidado. En esta ocasión hemos definido un array de punteros que apuntan a funciones que devuelven punteros definidos como tipo FLOAT.

Como se puede observar, las definiciones de punteros y arrays se pueden complicar todo lo que queramos (o podamos), aunque no es aconsejable abusar de este tipo de sintaxis en nuestras aplicaciones, puesto que después, a la hora de depurar los posibles (por no decir seguros) errores de funcionamiento que puedan aparecer, podremos llegar a perdernos y encontrarnos con que somos casi incapaces de saber qué hacía nuestro propio programa.

Realidad interna de una función

Saber la forma en que se almacenan los parámetros y demás datos en la PILA cuando llamamos y volvemos de una función en C/C++, en principio puede parecer que no es estrictamente necesario. Pero si deseamos realizar programas C que puedan llamar a procedimientos en ensamblador creados por nosotros mismos u otros que queramos añadir a un proyecto propio, es obligatorio saberlo para poder pasar los parámetros y que el procedimiento deseado los recoja de forma correcta de la pila. A continuación lo explicamos.

Antes de empezar la explicación, sólo decir que suponemos que estamos programando en 32 bits y compilando en el modo <parámetros pasados mediante pila> puesto que hay compiladores, como el Watcom, que aceptan también el pase de parámetros por registros y aunque es más rápido, es menos frecuente su utilización y menos estándar.

Cuando llamamos a una función y ésta resulta ser el procedimiento externo ensamblador que hemos definido, la PILA tiene los siguientes valores:

SS:ESP= Posición antigua Pila
SS:ESP+4= Parámetro 1.
SS:ESP+8= Parámetro 2.
SS:ESP+12= Parámetro 3.
etc...

Pasos a realizar en el procedimiento llamado:

1) Lo primero que tenemos que hacer con los registros es guardar el contenido del registro EBP en la PILA, para después dar el valor de la posición actual de la PILA a EBP o sea, **MOV EBP,ESP**.

2) Debemos guardar el contenido de todos los registros que vamos a usar dentro del procedimiento en la PILA para después restaurar sus contenidos antes de volver a la función llamadora.

3) Insertamos el código de aplicación y recordaremos que los parámetros se encuentran ahora almacenados desde la dirección **SS:[EBP+8]** en adelante, puesto que antes de dar a EBP el valor al que apuntaba ESP al llegar al procedimiento, este último fue guardado en PILA. Por lo que ahora, la pila está como sigue:

SS:EBP= reg. EBP salvado.
SS:EBP+4= Posición A.Pila.
SS:EBP+8= Parámetro 1.
SS:EBP+12= Parámetro 2.
SS:EBP+16= Parámetro 3.
etc...

Como se puede observar, tenemos que siempre, sean cuantos sean los parámetros que hayamos dado, y tras realizar el push EBP, tendremos localizado el primer parámetro en EBP+8.

4) Cuando termine el procedimiento, habrá que restaurar el estado de los registros afectados ejecutando la instrucción **LEAVE**, que libera el espacio ocupado por los parámetros y el *valor antiguo de Pila*, que fue almacenado en la función lla-

madora con una instrucción ensamblador tipo **ENTER**.

5) Para finalizar, simplemente ejecutar un **RET**, que retornará a la posición siguiente al **CALL** que llamó al procedimiento dentro de la función llamadora

Como complemento a todo lo explicado en la tabla 3 se ha incluido un ejemplo de un procedimiento ASM.

Tabla 3

```
;-----------------------------
; TABLA 3:
; Limpia_Buffer_Pantalla
; NºE: DD[COLOR] = Color de relleno.
; NºS: - No hay parámetros de salida
; NºD: Limpia el contenido del buffer
; de pantalla con el color
; de 32bits indicado.
; Tiempos: 486Dx50Mhz, VGA Cirrus
; logic vesa local bus.
; 1000 Borrados = 1.5s = 666.6b/s.
;-----------------------------
Limpia_Buffer_Pantalla proc near
 push ebp
 mov ebp,esp
 push edi

 cld
 mov ax,ds
 mov es,ax
 mov edi,VGA_BUFFER
 mov eax,[ebp+parm1]
 mov ecx,VGA_TAM_BUFFER
 shr ecx,2
 rep stosd

pop edi
leave
ret
Limpia_Buffer_Pantalla endp
```

Otras normas generales

En general se puede decir que todo programador que vaya a realizar programas serios y largos, debería respetar algunas normas básicas de programación y que son más o menos las que siguen:

1) Intentar limitarse siempre que sea posible al uso del ANSI-C, o sea, las funciones reconocidas como estándar C y evitar el uso de todas aquellas funciones propias del compilador. Aunque en principio pueda parecer innecesario, no podemos saber si algún día necesitaremos poder recompilar nuestra aplicación en otro entorno de desarrollo C/C++ o incluso peor todavía, en otra plataforma diferente (ej. pasar de PC a PowerPc). De esta forma, se ahorrará trabajo extra y los cambios necesarios que tendrá que realizar serán mínimos.

2) A la hora de realizar aplicaciones que requieren acceso a bajo nivel, a hardware y/o alta velocidad de proceso, y recurra al ensamblador, intente escribir la menor cantidad posible de código Asm posible y déjelo exclusivamente para rutinas puntuales como las de acceso masivo a pantalla o las de optimizaciones MMX específicas. De esta forma, si requiere más adelante convertir el programa a otra plataforma, la cantidad de código por reescribir será mucho menor. Por otra parte tener presente que los compiladores actuales consiguen niveles de optimización de código tan elevados como un programador experto en ensamblador, por lo que casi se hace innecesaria la programación a bajo nivel. Como curiosidad, sólo mencionar que algunas de las demos que se realizan en las famosas competiciones actuales, prácticamente ya no usan el ensamblador más que para alguna rutina de sonido o gráfica muy específica, obteniendo una proporción del 98% de código en C frente a un 2% de ensamblador.

3) Planificar el programa para que queden las funciones divididas en ficheros por capas, es decir, que queden las funciones en estratos según el nivel que tienen. Así, por ejemplo, el archivo con el main() principal, estará en el último nivel, situado en la capa más externa, y los de menor nivel, por ejemplo, los de acceso a archivos, gestión propia de memoria, funciones ASM y algoritmos básicos, estarán en el nivel más bajo. De esta forma, el programa queda más *clasificado*, es más fácil de modificar posteriormente y más legible.

4) Además de la clasificación por capas, deben separarse las funciones de cada nivel en ficheros según el tipo de funcionalidad. Así, por ejemplo, en una capa podemos tener un fichero con las funciones gráficas, otro con todos los algoritmos de cálculo 3D, etc...

Bibliografía

• Programación Orientada a objetos. Conceptos, modelado, diseño y codificación en C++. Luis Joyanes Aguilar. McGraw-Hill.

• C++ para programadores. Herbert Schildt. Osborne/McGraw-Hill.

• Programación en Microsoft C. Robert lafore, Grupo Waite. Anaya Multimedia.

• C++, Guía de autoenseñanza. Herbert Schildt. Osborne McGraw-Hill.

Programación de Packet Driver

Francisco José Abad Cerdá

En este artículo se explica qué son y cómo programar los Packet Driver, los cuales permiten un acceso a la red a muy bajo nivel.

Anteriormente a la aparición de los *Packet Driver*, los programas que hacían uso de una red local debían implementar sus propios *drivers* de red, de tal modo que el programador tenía que preocuparse no sólo de qué tipo de red soportaría el programa sino, además, de cuáles eran el fabricante y el modelo. Esto obligaba, ante un cambio en la tarjeta de red, a una recompilación del programa en la mayoría de los casos. Además tenía el inconveniente de que la aplicación, al ser la propietaria de la tarjeta, impedía a otras aplicaciones el uso de la misma.

La solución consiste en un pequeño módulo software que es el único que tiene acceso directo a la tarjeta de red y que ofrece un conjunto de servicios a todos los programas que se ejecutan en la máquina, permitiendo de esta forma a las aplicaciones desentenderse de la programación del hardware. Esto implica que puedan existir varias pilas de protocolos, que serán soportadas por un único *Packet Driver*, el cual se encargará de entregar los paquetes al protocolo que corresponda según lleguen a la tarjeta. Otra consecuencia es que el mismo programa puede funcionar en cualquier tipo de red, incluso en una que no existía en el momento de la programación (con ciertas restricciones). Esto es posible debido a que todos los *Packet Driver* ofrecen llamadas estándar que pueden usar las aplicaciones, independientemente de la tarjeta de red que lo soporte (puede ser una Ethernet, Token Ring, etc). Imaginémonos que existiera algo parecido en el terreno de las tarjetas de sonido: habría un *driver* específico para cada tarjeta, aportado por el fabricante, con un conjunto de llamadas estándar mediante las cuales se podría manejar la tarjeta. El mismo programa funcionaría sin cambios, ni configuraciones sobre una vieja Adlib, o sobre la última tarjeta con soporte de tabla de ondas, pudiendo además varios programas enviarle órdenes de forma concurrente.

Actualmente, la mayoría de los fabricantes de hardware de red incorporan, junto al resto de *drivers* para su tarjeta, el *Packet Driver*. También hay grupos de usuarios o empresas que ofrecen *Packet Driver* para distintas tarjetas de modo gratuito. Por ejemplo, la colección Crynwr se encuentra en el directorio /msdos/pktdrvr en cualquier *mirror* de SimTel (p.e. ftp.funet.fi). Por supuesto, siempre que dispongamos del *driver* del fabricante, es aconsejable usarlo antes que cualquier otro. Aunque en la máquina existan varios programas activos a la vez, cada uno está atendiendo a un cierto tipo de paquetes, ignorando el resto. Para ello, la aplicación debe avisar al *Packet Driver* para que le entregue un cierto tipo de paquetes (se dice que se 'registra'). En cuanto el programa acabe, debe avisar al *Packet Driver* para que no le envíe más paquetes. De igual modo, se facilita la programación en los casos en que existen varias tarjetas en un mismo PC, como por ejemplo, un puente, que sirve para unir redes locales y para 'filtrar' paquetes de una a otra. En este caso se aconseja que haya un PD distinto por cada tarjeta. Tan sólo habría

que recoger el paquete que entra por una tarjeta, comprobar que tiene permiso para atravesar el puente y retransmitirlo por la otra tarjeta. Si no tiene permiso, simplemente se descarta.

Hay tres niveles de *Packet Driver*, que se distinguen según las funciones que ofrecen:

Básico: ofrece la mínima funcionalidad para enviar y recibir paquetes. Utiliza el mínimo de recursos de la máquina.

Extendido: engloba al básico, añadiendo funciones que se usan menos, tales como *multicast* y ofrecimiento de estadísticas de uso del interfaz

Altas prestaciones: pueden acompañar al PD básico o al extendido y soporta funciones de mejora de prestaciones y ajuste.

Antes de llamar a funciones extendidas o de altas prestaciones, debemos comprobar que el *driver* de nuestra tarjeta los soporta, ya que es usual que no todos los *drivers* implementen todos los niveles (normalmente soportan el básico y el extendido). Esto se puede consultar mediante una función, que devuelve el tipo del *Packet Driver*.

Ethernet

El estándar Ethernet es el tipo de red local más extendido debido a la relación precio/prestaciones que ofrece. Actualmente, se puede montar una red local a 10 Mbps por alrededor de 5.000 pesetas por ordenador.El formato de los paquetes que circulan por una Ethernet aparece en la figura 1.

El destino y el origen son direcciones de 6 bytes que están grabadas normalmente en la ROM de la tarjeta y son únicas entre todas las tarjetas del mundo. Esto se consigue asignando una parte de la dirección a cada fabricante, encargándose éste de generar números únicos con el resto de la dirección. Además, la dirección FF:FF:FF:FF:FF:FF en hexadecimal representa un *broadcast*, es decir, el paquete que lleve esta dirección en el campo destino llegará a todos los ordenadores de la red. También hay un método denominado *multicast* por el que la trama llegará a un grupo de ordenadores y que se explicará luego. El preámbulo y el CRC los genera la tarjeta automáticamente, por lo que no nos debemos de preocupar por ellos. El campo *tipo* es el que permite al *Packet Driver* discriminar la aplicación a la que va destinada cada trama.

Programación de Packet Driver

El *Packet Driver* se encuentra en una interrupción entre la 0x60 y 0x80 (en notación de C). Se le permite un rango de interrupciones, en vez de asignarle una fija para reducir riesgos de conflicto con otro software existente y para que se pueda instalar más de un *Packet Driver*. Así, una aplicación que desee usar la red debe buscar la firma entre las direcciones a las que apuntan los vectores indicados, esta firma deberá ser "PKT DRVR". Por ejemplo, si se quiere ver si hay un *Packet Driver* instalado en el vector de interrupción 0x60, se debe leer la dirección a la que apunta ese vector, con una llamada del tipo *getvect(0x60)*. Luego, a esa dirección se le suma 3 y en esa dirección debe aparecer la firma indicada. Esos tres bytes se corresponden con un salto a la zona de código del *Packet Driver*. Cada tarjeta en un ordenador se puede identificar mediante una tripleta de 3 números. El primero, llamado *clase*, de 8 bits identifica el tipo de red (Ethernet, Token Ring...). El segundo, llamado *tipo*, es un entero de 16 bits que identifica al fabricante de la tarjeta. El último es el *número* de interfaz (8 bits), que distingue a dos o más tarjetas en el caso de que haya más de una del mismo tipo.

Todas las operaciones que ofrece el *Packet Driver* se llaman mediante la interrupción software que se ha buscado antes, del mismo modo que se hace con las llamadas a la BIOS o al DOS. El código de operación se le pasa en el registro AH y los demás parámetros de cada función también se pasan en registros del procesador. El error se indica mediante la bandera de acarreo del procesador y mediante el registro DH cuando acaba la función.

Como se ha dicho antes, debe haber un modo de distinguir para cada paquete que llega a una cierta máquina, a qué pila de protocolos va dirigido. En Ethernet esto se consigue mediante el campo 'tipo' del paquete. El programa, mediante una llamada a la función **access_type**, le indica al driver que quiere obtener un tipo determinado de paquetes. En esa llamada se le devuelve un *handle* (manejador), que se almacena en un entero y que identificará todas sus operaciones posteriores relacionadas con ese tipo de paquetes. Es obligatorio que, cuando la aplicación acabe, llame a la función **release_type**, que indica al *Packet Driver* que ya no quiere más paquetes de ese tipo.

Antes de entrar a describir las funciones hay que tener en cuenta el orden de almacenamiento en memoria de los enteros. El PC guarda primero el byte menos significativo y luego el más significativo, los diseñadores de la red optaron por la otra elección. Así, todos los paquetes que circulan por la red, cuando llevan algún entero, lo llevan en orden contrario a como

Figura 1

Preámbulo	Destino	Origen	Tipo	Datos	CRC
8 bytes	6 bytes	6 bytes	2 bytes	46-1500	4 bytes

Tabla 1. driver_info.

Entrada	Error (CF=1)	Correcto (CF=0)
AH=1	DH=codigo de error	BX=versión
AL=255		CH=clase
BX=handle		DX=tipo
		CL=número
		DS:DI= nombre
		AL=funcionalidad

los PC lo almacenan en memoria. Esto obliga a que, al hacer alguna llamada que implique un entero, se deba invertir el orden de sus bytes.

Funciones básicas

driver_info(int handle)

Esta función devuelve información sobre el *Packet Driver* instalado, que se necesitará luego en la llamada a **access_type**. También informa sobre la funcionalidad del *Packet Driver* en el registro AL, según los siguientes valores:

1 = funciones básicas
2 = funciones básicas y extendidas
5 = básicas y de altas prestaciones
6 = básicas, extendidas y de altas prestaciones
255 = no instalado

El parámetro *handle* es opcional y se puede dejar a cero. Ver tabla 1.

access_type(unsigned char clase,
int tipo,
unsigned char numero,
char far *tipop,
int long_tipo,
int (far *receptor)())

Esta función es una de las más importantes del *driver*, ya que inicia la captura de un determinado tipo de paquetes. Los parámetros *clase*, *tipo* y *numero* se refieren a los valores devueltos por la llamada a **driver_info**. El argumento *tipop* es un puntero al tipo de tramas que se desea y *long_tipo* es la longitud del tipo anterior. En Ethernet, la longitud siempre es 2. ¡Ojo con el orden de los bytes si se mantiene el tipo almacenado en un entero! Si se hace así, se deben intercambiar los bytes antes de llamar a la función. Si se le pasa como longitud del tipo 0, entonces el *Packet Driver* entregará todos los paquetes a la aplicación, independientemente de su clase. El parámetro *receptor* es una subrutina que le debe pasar la aplicación, que se llamará cada vez que le llegue un paquete. Esta subrutina se llama dos veces.

Tabla 2. access_type.

Entrada	Error (CF=1)	Correcto (CF=0)
AH=2	DH=codigo de error	AX=handle
AL=clase		
BX=tipo		
DL=número		
DS:SI=tipop		
CX=long_tipo		
ES:DI=receptor		

La primera vez el registro AX está a 0 y pide un búfer para almacenar el paquete en los registros ES:DI. El tamaño del búfer viene dado en CX, e incluye la cabecera (dirección destino, origen y tipo). En BX está el handle. Si se le devuelve 0:0 como dirección, el Packet Driver descartará el paquete. En la segunda llamada, AX está a 1, e indica que la copia está completa. En DS:SI está almacenada la dirección donde se ha copiado el búfer. Ver tabla 2.

release_type(int handle)

Esta llamada se encarga de avisar al *Packet Driver* de que deje de llamar a la función 'receptor' ante la llegada de un determinado tipo de paquetes, ya que el programa ha acabado. Es muy importante que se llame a esta función antes de acabar, puesto que si no, el Packet Driver seguiría saltando a la dirección de memoria indicada en **access_type**, sin saber lo que puede haber allí al acabar el programa, lo que normalmente acaba en un cuelgue de la máquina. Ver tabla 3.

send_pkt(char far *buffer, unsigned longitud)

Pone en la red el paquete que se encuentra en la dirección *buffer* cuyo tamaño viene dado por *longitud*. La aplicación debe poner el paquete entero, es decir, la dirección destino, origen, tipo y los datos para el caso de Ethernet. En este punto, se debe tener en cuenta que si dentro del paquete se transmiten enteros, éstos deben estar en el formato de almacenamiento de la red (primero el byte más significativo). Esto es especialmente importante cuando se van a comunicar un PC con una máquina no PC, ya que se asume que todos los enteros que circulen por la red tendrán ese formato. Ver tabla 4.

terminate(int handle)

Esta es una llamada que no se usa mucho. Finaliza el *Packet Driver* asociado con el *handle* y, si puede, lo descarga de la memoria. Ver tabla 5.

Tabla 3. release_type.

Entrada	Error (CF=1)	Correcto (CF=0)
AH=3 BX=handle	DH=codigo de error	

Tabla 4. send_pkt.

Entrada	Error (CF=1)	Correcto (CF=0)
AH=4 DS:SI= búfer CX+longitud	DH=codigo de error	

Tabla 5. terminate.

Entrada	Error (CF=1)	Correcto (CF=0)
AH=5 BX=handle	DH=código de error	

get_address(int handle, char far *buffer, int longitud)

Devuelve la dirección de la tarjeta asociada al *handle* indicado. Se debe tener en cuenta que esta dirección puede no ser la dirección grabada en ROM, ya que ésta se puede cambiar con una llamada a **set_address**. El *buffer* es un puntero largo a la zona de memoria donde se quiere almacenar la dirección y con la *longitud* se le indica el tamaño de ese búfer. Al acabar la llamada, en CX se devuelve el tamaño realmente usado en el búfer. Ver tabla 6.

reset_interface(int handle)

Resetea el interfaz asociado al *handle*, abortando cualquier transmisión en curso y reinicializando el receptor. La dirección de red se vuelve a leer de la ROM, la lista de multicast se limpia y se activa el modo normal de recepción (sólo se reciben los paquetes dirigidos a la máquina y los 'broadcast'). Sólo debe haber un *handle* activo. Ver tabla 7.

Funciones extendidas

set_rcv_mode(int handle, int modo)

Establece el modo de recepción, según los siguientes valores:

1 apaga el receptor
2 recibir sólo paquetes dirigidos a esta tarjeta
3 modo 2 más los paquetes broadcast
4 modo 3 más los paquetes multicast limitados
5 modo 3 más todos los paquetes multicast
6 todos los paquetes

Todos los interfaces no tienen por qué implementar la totalidad de los modos. Como la función es extendida, incluso puede haber *Packet Driver* que no soporten la llamada. Si esto ocurriese, se presupone el modo 3, que es el normal. Cuando se cambia el modo, aunque se ponga un *handle* determinado, se está cambiando el modo del interfaz completo, así que ¡cuidado cuando hay más de una aplicación en marcha! Ver tabla 8.

get_rcv_mode(int handle)

Devuelve el modo de recepción activo de la tarjeta asociada al *handle*. Ver tabla 9.

set_multicast_list(char far *listadir,int longitud)

El parámetro *listadir* apunta a una lista de direcciones multicast de *longitud* bytes. Las direcciones multicast se encargan de enviar un paquete a un conjunto de máquinas definido. El broadcast llega a todas las máquinas conectadas a la red. Como el hardware tiene un búfer limitado de direcciones multicast, se recomienda salvar la lista original de direcciones multicast, con la llamada **get_multicast_list** para poder restaurarla luego en caso de error o cuando termine la aplicación. Ver tabla 10.

get_multicast_list()

La función devuelve un puntero a la lista de direcciones multicast activa y su tamaño. Esta lista no se debe modificar directamente. Ver tabla 11.

Tabla 6. get_address.

Entrada	Error (CF=1)	Correcto (CF=0)
AH=6 BX=handle ES:DI=búfer CX=longitud	DH=codigo de error	CX=longitud

get_statistics(int handle)

Devuelve un puntero a una estructura como la que se muestra a continuación, conteniendo las estadísticas de uso de la tarjeta. Los valores se almacenan en formato Intel, por lo que se pueden usar directamente. Ver tabla 12.

```
struct estadisticas{
// Paquetes recibidos en todos los handles
        unsigned long recibidos;
// Paquetes mandados
        unsigned long transmitidos;
// Incluyen las cabeceras
        unsigned long bytes_recibidos;
        unsigned long bytes_transmitidos;
// Incluyen todos los tipos de error
        unsigned long errores_in;
        unsigned long errores_out;
// Sin espacio en la cola, tarjeta colapsada, etc
        unsigned long perdidos;
}
```

set_address(char far *direccion, int longitud)

Esta llamada 'cambia' la dirección hardware de la tarjeta, por aquella apuntada por el parámetro *direccion*, de *longitud* bytes (en Ethernet siempre 6). Esta llamada se usa a veces en algunas aplicaciones o protocolos, que necesitan una dirección específica de red. Sólo debe haber un *handle* abierto. Ver tabla 13.

Funciones de altas prestaciones

get_parameters()

Una aplicación puede mejorar el uso de la tarjeta conociendo los parámetros ofrecidos por esta función. La función devuelve un puntero a una estructura como la siguiente:

```
struct parametros {
        unsigned char rev_mayor;
        unsigned char rev_menor;
        unsigned char longitud;
        unsigned char long_dir;
        unsigned short mtu;
        unsigned short tam_multicast;
        unsigned short buf_recep;
        unsigned short buf_trans;
        unsigned short num_int;
}
```

Los parámetros *rev_mayor* y *rev_menor* indican la revisión del *driver*. Por ejemplo, este artículo se ajusta a la especificación 1.09, por lo que *rev_mayor* sería 1 y *rev_menor* sería 9. La *longitud* contiene el tamaño de la estructura. Esto se usa para poder ampliarse en futuras revisiones. Para la que se está usando en este artículo, vale 14. El parámetro *long_dir* contiene la longitud de las direcciones que usa la tarjeta. El *mtu* es el tamaño del paquete más grande que puede manejar la tarjeta, incluyendo la cabecera. En caso de Ethernet este tamaño es fijo, de 1514 (1500 bytes de datos mas 14 de cabecera), en otro tipo de redes, el tamaño puede no ser fijo. *tam_multicast* indica el tamaño del búfer interno de direcciones multicast de la tarjeta. Un cero en este campo indica que no se soporta multicast. Los parámetros *buf_recep* y *buf_trans* indican el número de paquetes menos uno que puede retener la tarjeta en recepción o en transmisión. Valores bajos en estos parámetros quieren indicar a las aplicaciones que implementen algún protocolo de control de flujo, ya que si llegan muchos paquetes a la tarjeta, normalmente se colapsará y empezará a descartar paquetes. *num_int* apunta a una interrupción hardware a la que se puede enganchar la aplicación para hacer algún tratamiento inmediatamente después del fin de interrupción de la

Tabla 7. reset_interface.

Entrada	Error (CF=1)	Correcto (CF=0)
AH=7 BX=handle	DH=código de error	

Tabla 8. set_rcv_mode.

Entrada	Error (CF=1)	Correcto (CF=0)
AH=20 BX=handler CX=modo	DH=codigo de error	

Tabla 9. get_rcv_mode.

Entrada	Error (CF=1)	Correcto (CF=0)
AH=21 BX=handle	DH=código de error	AX=modo

Tabla 10. set_multicast_list.

Entrada	Error (CF=1)	Correcto (CF=0)
AH=22 ES:DI=listadir CX=longitud	DH=código de error	

Tabla 11. get_multicast_list.

Entrada	Error (CF=1)	Correcto (CF=0)
AH=23	DH=código de error	ES:DI=lisadir CX=longitud

Tabla 12. get_statistics.

Entrada	Error (CF=1)	Correcto (CF=0)
AH=24 BX=handle	DH=código de error	ES:DI=estadistica

Tabla 13. set_address.

Entrada	Error (CF=1)	Correcto (CF=0)
AH=25 ES:DI=dirección CX=longitud	DH=código de error	CX=longitud

Tabla 14. get_parameters.

Entrada	Error (CF=1)	Correcto (CF=0)
AH=10	DH=código de error	ES:DI=parametros

Tabla 15. as_send_pkt.

Entrada	Error (CF=1)	Correcto (CF=0)
AH=11 DS:SI=búfer CX=longuitud ES:DI=aviso	DH=código de error	

tarjeta. Para usar esta opción hay que ser cuidadoso, puesto que debe ser reentrante. Ver tabla 14.

```
as_send_pkt(char far *buffer,
    unsigned longitud,
    int (far *aviso)())
```

A diferencia de la llamada **send_pkt**, esta llamada no es bloqueante. Esto quiere decir que no se espera a que termine la transmisión, sino que devuelve el flujo de ejecución inmediatamente a la aplicación. Así, no se puede modificar el búfer hasta recibir el *aviso*, lo que indicará que se ha copiado el paquete en un búfer interno de la tarjeta, por lo que ya se puede modificar. Nótese que el paquete puede no haberse mandado aún. Si ha habido algún error en la llamada, no se recibirá el aviso. Dentro de la llamada al aviso, se tiene:

AX=0 si se ha copiado bien
ES:DI= dirección del búfer que se indicó en la llamada a 'as_send_pkt'. Ver tabla 15.

El la tabla 16 aparecen todos los códigos de error que puede devolver el Packet Driver.

Junto al artículo se ofrece una librería en C que implementa todas las llamadas al *Packet Driver*, para no tener que preocuparse de registros, interrupciones, etc. Además se acompaña de una pequeña utilidad de chat, que permite 'hablar' entre dos estaciones de la misma red.

Tabla 16. Códigos de error.

1.- Número de handle no válido
2.- No se ha encontrado un interfaz con esa clase
3.- No se ha encontrado un interfaz con ese tipo
4.- No se ha encontrado interfaz con ese número
5.- Tipo de paquete incorrecto
6.- Este interfaz no soporta multicast
7.- Este Packet Driver no se puede desinstalar
8.- Se ha especificado un modo de recepción no válido
9.- No hay suficiente espacio
10.- El tipo se estaba usando y no ha sido liberado
11.- Comando fuera de rango o no implementado
12.- No se puede mandar el paquete (normalmente fallo hardware)
13.- No se puede cambiar la dirección hardware porque hay más de un handle abierto
14.- Dirección hardware incorrecta (longitud o formato erróneos)
15.- No se puede reinicializar el interfaz porque hay más de un handle abierto

Criptografía. De la antigüedad al DES.

José Antonio Mendoza

El arte de la criptografía, antes recluido en el mundo diplomático y militar y en manos de unos pocos sabios, se ha popularizado y difundido de una manera inimaginable hace tan solo unos años, debido a la revolución tecnológica de los últimos años del siglo XX.

El hombre, desde siempre, ha intentado proteger su ciencia, sus conocimientos, sus recursos naturales; en resumen lo que le diferenciaba de otras colectividades humanas, dotándole de una superioridad sobre las demás. Con la invención de la escritura nació la necesidad de comunicar escritos de una manera segura, lo que implicaba audaces correos (canales seguros), sobres lacrados y sellados (autentificación, no adulteración del mensaje, seguridad de que nadie, excepto el destinatario, lo leyera, etc.).

Criptografía

Antes de comentar los métodos criptográficos tenemos que definir los términos que intervienen en una comunicación de esta naturaleza.

Texto en claro es aquel que queremos convertir en un mensaje ininteligible para cualquier persona excepto para el emisor y receptor.

Texto cifrado es el texto obtenido después de cifrar.

Cifrador es el método o mecanismo por el que convertimos el texto en claro en texto cifrado.

Canal es el medio por el cual va a viajar nuestro mensaje, con la particularidad de que en algún tramo, o en su integridad, puede ser escuchado por criptoanalistas -personas dedicadas a intentar descifrar un mensaje- partiendo de no toda la información necesaria para desencriptarlos.

Descifrador es el método o mecanismo por el que convertimos el texto cifrado en texto en claro.

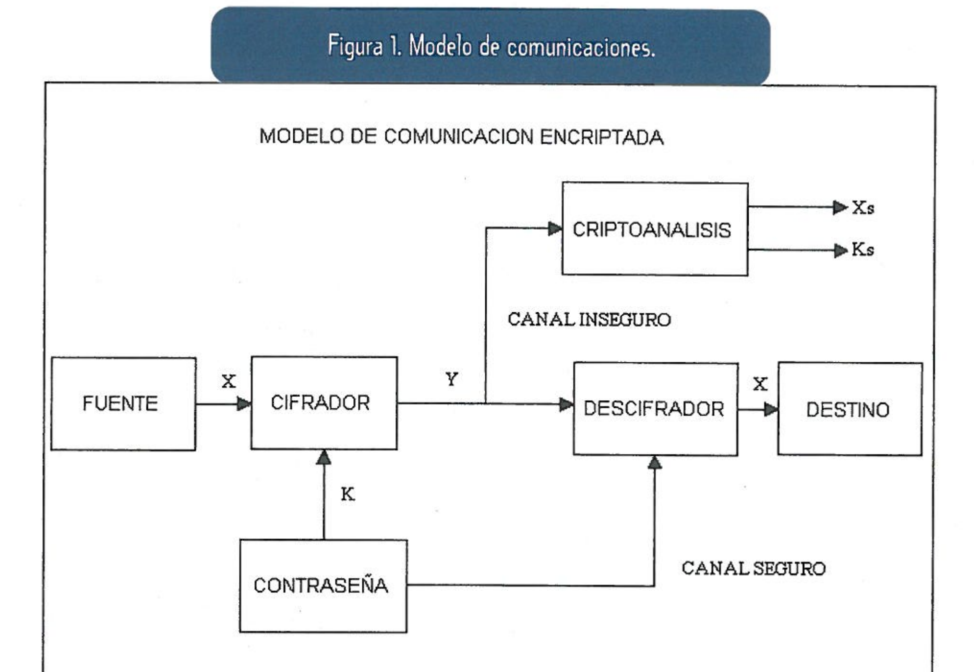
Contraseña, *password,* clave, etc. Información compartida por el emisor y receptor mediante la cual es posible cifrar y descifrar el mensaje. Lógicamente el emisor y receptor también deben compartir el método de cifrado (ver figura 1).

Texto en claro es aquel que queremos convertir en un mensaje ininteligible para cualquier persona excepto para el emisor y receptor

Llegados a este extremo nos debemos preguntar si, elegido un método de cifrado, nuestra comunicación es segura. Existen dos tipos de seguridad: la incondi-

cionalmente segura, que consiste en que el texto cifrado no contiene información suficiente para que a partir de él se deduzca el texto en claro, sin importar el esfuerzo y los medios ilimitados que se empleen. Y por otra parte, el cifrado computacionalmente seguro, el cual se produce debido a que el coste de averiguar el mensaje es superior al valor de la información contenida en el texto cifrado y a que el tiempo requerido para descifrar el mensaje es superior a la validez temporal de dicho mensaje.

Figura 1. Modelo de comunicaciones.



Cifradores clásicos

El método de cifrado más simple es el esteganogáfico y, ¿qué significa este término tan extraño? Pues se traduce por algo tan simple como ocultar el mensaje dentro de otro de apariencia intranscendente. A lo largo de la historia ha habido muchos, a cada cual más curioso. De los más interesantes tenemos los siguientes: remarcar caracteres en un texto, por ejemplo con tinta más brillante, lo que con luz intensa hace resaltar al texto en clave. Otra manera es con tinta invisible, como el zumo de limón que al calentar el papel aparece el mensaje. También posicionando las palabras del mensaje, según una regla, dentro del mensaje intranscendente. Por ejemplo en la tercera palabra de cada línea del texto.

Las dos técnicas clásicas de cifrado de textos son la sustitución de caracteres siguiendo una determinada regla y la transposición de caracteres

Esto es lo que se hacía en la antigüedad, pero ahora con los ordenadores se pueden esconder mensajes en el interior, por ejemplo, de fotos en formato digital; basta alterar algunos bits, los menos significativos, que en poco, o en casi nada, cambian el aspecto de la misma, para introducir nuestro mensaje oculto.

Las dos técnicas clásicas de cifrado de textos son la sustitución de caracteres siguiendo una determinada regla y la transposición de caracteres, o sea la alteración del orden de los caracteres del mensaje.

Métodos de sustitución

Julio Cesar utilizó un método de cifrado, al que ha dado nombre. Consiste en sustituir unas letras por otras según una determinada regla, que consiste en un desplazamiento del alfabeto, es decir, en el renglón de arriba ponemos el alfabeto normal: abcdefghi y debajo aparecería también pero con la particularidad de poner por ejemplo la a debajo de la f. De esta manera haremos corresponder la f con la a, la g con la b la h con la c, etc., y así sucesivamente con las letras del alfabeto. Ya tenemos la regla de sustitución de unos caracteres por otros. Por ejemplo, la frase "SOLO PROGRAMADORES" se escribiría de forma encriptada: "ÑKGK LNKBNUHUYKNZÑ". Aunque a primera vista nos parecería difícil descifrar el texto, si conocemos el método de encriptado sólo tendremos que probar 26 desplazamientos distintos y tendremos el texto en claro solo a partir del texto cifrado sin saber la correspondencia inicial.

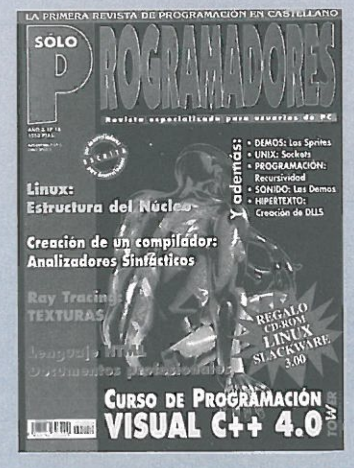
El cifrado Cesar consiste en sustituir unas letras por otras según una regla que consiste en un desplazamiento del alfabeto

números atrasados

PROGRAMADORES

completa ya tu colección



























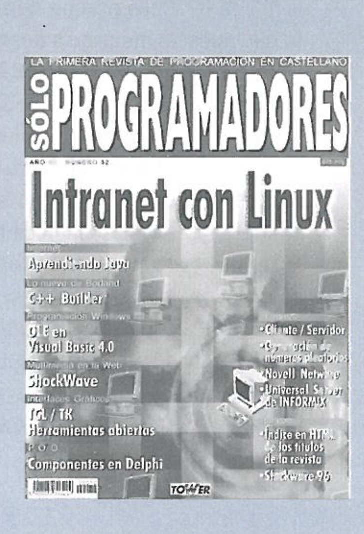
LA PRIMERA REVISTA DE PROGRAMACIÓN EN CASTELLANO
SÓLO PROGRAMADORES
BORLAND INTRABUILDER PROFESIONAL 1.0
COMUNICACIONES EN WINDOWS
PROGRAMACIÓN DE INTERFACES GRÁFICAS CON WINDOWS X



LA PRIMERA REVISTA DE PROGRAMACIÓN EN CASTELLANO
SÓLO PROGRAMADORES
PROGRAMACIÓN EN JAVASCRIPT
GESTIÓN DE EVENTOS • SCRIPTS INCRUSTADOS
COMPARATIVA JAVA, CGI
MULTITAREA EN TAPI
DEMOSCENE: MODELOS DE ILUMINACIÓN
BORLAND C++ 5.01



LA PRIMERA REVISTA DE PROGRAMACIÓN EN CASTELLANO
SÓLO PROGRAMADORES
NIVEL DE USUARIO EN LINUX
DEMOSCENE: INCLUYENDO OBJETOS 3DS
SP
GESTIÓN DE PROTOCOLOS
NETWARE Novell



LA PRIMERA REVISTA DE PROGRAMACIÓN EN CASTELLANO
SÓLO PROGRAMADORES
Intranet con Linux
Aprendiendo Java
C++ Builder
OLE en Visual Basic 4.0
ShockWave
TCL / TK
Herramientas abiertas
Componentes en Delphi

El cifrado Cesar deja mucho que desear; para mejorarlo se utilizó otra regla de correspondencia de unas letras con otras. El Cifrado monoalfabético sólo cambia respecto al Cesar en que en vez de hacer un desplazamiento del alfabeto, ahora sustituiremos una letra por otra, unívocamente, sin ninguna relación de orden. El incremento de posibilidades en la correspondencia entre las letras es abismal, pasando de tener 26 a 26 !.

Sin embargo, aunque parezca mentira, descifrar a partir sólo del texto cifrado el texto en claro es trivial. La clave está en la frecuencia de aparición de las letras según el idioma que empleemos. Cada idioma tiene su firma, es decir la frecuencia de aparición de las letras en un texto no es aleatoria. Basta con un mensaje de unas cuantas frases para, a partir de la frecuencia de aparición de las letras, identificar la correspondencia de las letras en claro con las cifradas. Para entenderlo mejor, basta con describir el proceso. Contamos las apariciones de cada letra en el mensaje cifrado. La que más veces aparezca la emparejamos con la E, la segunda con la T, la tercera con la R y así sucesivamente. ¿Por qué con la E, la T y la R? Pues porque hemos supuesto que nuestro mensaje a descifrar estaba escrito en inglés y en inglés la letra más frecuente es la E le sigue la T después la R y así sucesivamente. Es posible que no obtengamos el texto en claro a la primera, debido a que el texto no sea completamente fiel a las probabilidades del idioma al que pertenece, pero tendremos un número de letras acertadas suficientes como para a partir de contexto encontrar la correspondencia de las que nos queden. Cada idioma tiene su plantilla de probabilidades, su firma, que lo identifica unívocamente.

El cifrado polialfabético consiste en una mejora del Cesar

Llegados a este punto, el objetivo se encontraba en alterar de alguna manera la frecuencia de aparición de las letras, que a partir de las frecuencias de aparición, no se pudiera deducir la correspondencia de los dos alfabetos. Para lograrlo se intentaron fundamentalmente dos métodos, uno fue pasar de cifrar letra a letra, a cifrar parejas de letras, cada pareja de letras tendrá su pareja cifrada correspondiente, la otra fue el cifrado polialfabético.

Cifrado de conjuntos de letras

En el cifrado de conjuntos de letras, éstos estaban formados por dos letras, aunque el método se puede extender a conjuntos de más letras. En su época supuso un gran avance pues difuminaba bastante la firma de un idioma, no pudiendo desencriptar con un simple ojeo de probabilidades. De hecho este método fue considerado seguro hasta la segunda guerra mundial, donde fue ampliamente utilizado por el ejército americano. Su nombre se debe a Playfair, quien lo introdujo en el Ministerio de asuntos exteriores Británico a mediados del siglo pasado. Básicamente consiste en formar una matriz de cinco por cinco, tendríamos en nuestro alfabeto por tanto veinticinco letras. Para pasar de veintiséis letras del alfabeto inglés, a veinticinco, se agrupan en un elemento de la matriz dos de ellas, siempre que estén seguidas y preferentemente sean poco probables en el idioma usado. ¿Cómo formamos dicha matriz ?. Basta con elegir nuestra palabra clave, en la cual no se pueden repetir letras. Dicha palabra clave la pondríamos de izquierda a derecha, en la primera fila de la matriz. Si faltasen casillas, continuaríamos por la segunda fila y así sucesivamente. Una vez puesta la clave completaríamos con las letras que no estuvieran en dicha clave, siempre de izquierda a derecha y de arriba a abajo.

Describiremos ahora el algoritmo de cifrado que sigue las siguientes reglas: cada letra se hacía corresponder con otra de su misma fila, aquella que estuviera en la misma columna que la otra letra de la pareja. Es posible que dos letras coincidan en la misma fila, entonces les corresponden las letras que tengan a su derecha; si una de ellas estuviera al final de la fila, por la derecha, le correspondería la primera de su fila por la izquierda. Asimismo si coinciden en la misma columna le correspondería la letra que tienen debajo, si una de ellas estuviera en la última fila le correspondería la primera letra de la columna por arriba. Sólo quedan dos excepciones, una es cuando tenemos dos letras seguidas iguales, en ese caso introducimos en el texto a cifrar una letra de relleno entre las dos letras repetidas, convenida ésta entre emisor y receptor. La otra particularidad se trata de la casilla donde tenemos dos letras, en ese caso ciframos indistintamente por una de ellas, el contexto nos llevará a descifrar correctamente. Para descifrar un texto construiríamos la matriz de la misma forma y los conjuntos de letras cifradas nos llevarían, siguiendo las mismas reglas en sentido inverso, a las parejas de texto en claro.

A pesar de la robustez de este método, no logra eliminar las redundancias de un idioma. Terminaciones como mente, ando, ion, artículos como la, el, raíces de verbos, así como sus terminaciones temporales, son grupos de letras que una vez codificados, nos dan pistas para establecer correspondencias entre letras y por último encontrar la matriz. Gracias a los ordenadores estos métodos han quedado obsoletos.

Cifrado Polialfabético

Nos queda el cifrado polialfabético que consiste en una mejora del Cesar. Construimos una matriz de veintiséis por veintiséis, es decir, todos los desplazamientos del cifrado Cesar uno debajo del

otro: la primera fila ABCDEFYZ , la segunda BCDEFYZA, y así sucesivamente. Para encriptar, ponemos arriba la clave muchas veces ; claveclaveclave tantas como sean necesarias y debajo nuestro texto en claro soloprogramadores, la c y la s nos llevan a la letra correspondiente, en este caso la u, la l con la o a z, así cifraríamos todo nuestro texto, como si jugáramos a los barcos. Este método se conoce como de Vigenère. Como no podía ser menos, este método también presenta debilidades pudiendo ser descifrado. Éstas vienen de la periodicidad, es decir, como la clave tiene una longitud finita cuando una secuencia de letras que codificar va apareciendo en el texto, la misma secuencia codificada también aparece en el texto cifrado periódicamente, de esta manera se puede determinar la longitud de la clave, una vez conocida ésta se empieza a buscar coincidencias entre letra de texto en claro y letra de texto cifrado. La manera de protegerse de esta debilidad consiste en alargar la clave y en su caso extremo hacerla tan larga como el texto a cifrar. Pero ni aún así, ya que al estar la clave escrita en un determinado idioma sus probabilidades aparecerían en el texto cifrado, por lo tanto aprovechando esa debilidad se podría desencriptar. Únicamente generando una clave aleatoria de la misma longitud que el texto no dispondríamos de información suficiente para desencriptarlo.

Las máquinas de rotores fueron ampliamente utilizadas por los alemanes y japoneses durante la segunda guerra mundial

El problema de este método está en que emisor y receptor deben compartir la misma secuencia aleatoria, con los problemas de seguridad que eso conlleva, así como que no puede ser usada muchas veces seguidas debido a que se podrían encontrar correlaciones entre unos mensajes y otros. Una secuencia aleatoria consiste en que todas las letras que aparecen lo hacen con la misma probabilidad y con independencia, es decir, que tras la aparición de unos caracteres no se pueda predecir la aparición de otros.

Máquinas de rotores

El siguiente paso en los algoritmos de sustitución fueron las máquinas de rotores, ampliamente utilizadas por los alemanes y japoneses durante la segunda guerra mundial, los primeros tenían la *enigma* y los segundos la *púrpura*, que estaban basadas en sustituciones polialfabéticas. Durante el conflicto los aliados fueron capaces de desencriptar los mensajes generados por dichas máquinas. La ventaja estratégica conseguida fue uno de los factores que decantaron la victoria aliada sobre el eje. Veamos cómo funcionan dichas máquinas.

Una máquina de rotores tiene un funcionamiento simple. Suelen constar de una serie de cilindros, "rotores", ya que éstos giran según se van enviando caracteres. Cada cilindro tiene veintiséis contactos de entrada por una cara, y otros tantos de salida por la otra. Los contactos de salida y entrada están unidos unívocamente en el interior del cilindro, es decir, hay una conexión entre un contacto de entrada y otro de salida. Por lo tanto tenemos una serie de cilindros insertados en un eje, los contactos de salida de un cilindro se tocan con los de entrada del siguiente, de tal manera que los impulsos eléctricos pasan de un cilindro a otro según un camino determinado por las diferentes conexiones posibles entre los distintos cilindros. La potencia del método consiste en que por cada envío de un carácter, el cilindro situado más a la derecha gira una posición y cuando completa una vuelta, gira una posición el cilindro que le precede y así hasta llegar al cilindro que se encuentra en primer lugar. Mediante este proceso se obtiene una sustitución múltiple polialfabética, tantas como el número de cilindros.

Imaginemos que queremos realizar una transmisión cifrada. Primero emisor y receptor se tienen que poner de acuerdo en el juego y posiciones de los cilindros a utilizar. Luego hay que sincronizar los cilindros. Entonces ya se puede empezar a transmitir. Al pulsar una letra se hace circular una corriente por el contacto asociado a dicha letra del primer cilindro. Esa corriente pasa por las diferentes conexiones de los distintos cilindros, terminando en el contacto de salida del último cilindro, enviándose al medio el carácter asociado a este último carácter. A su vez se hace rotar una posición al último de los cilindros. Cuando se complete una vuelta de éste, el penúltimo giraría una posición, y así con todos ellos. En recepción se operaría de la misma forma, solo que la corriente excita una bobina, que mueve un electroimán unido al percutor de una máquina de escribir con la letra asociada.

Estos sistemas tienen veintiséis por n sustituciones alfabéticas posibles, donde n es el número de rotores de que conste dicha máquina, siendo ese número el periodo de repetición. Como vimos antes, si logramos identificar el periodo de repetición de una clave, ya tenemos prácticamente hecho el criptoanálisis.

Hemos visto la evolución de los métodos de sustitución, desde su inicio conocido hasta que dejaron de ser operativos al término de la segunda guerra mundial, debido fundamentalmente a los potentes métodos informáticos. Vamos a ver ahora el otro método clásico, la transposición de letras del texto en claro.

Métodos de transposición

Conceptualmente simples, las técnicas de transposición consisten básicamente en alterar el orden de las letras de

un texto siguiendo una determinada regla, conocida por emisor y receptor. Estas técnicas podrían ir desde poner primero las letras pares del mensaje y después las impares, hasta realizar múltiples transposiciones encadenadas.

Lo primero que podemos hacer es escribir el texto en filas, formando una matriz, para alterar el orden de las letras del texto en claro, basta transmitir las columnas en un orden, según una clave numérica. Tras esta operación, el texto cifrado no ha perdido las propiedades probabilísticas del idioma del texto en claro, lógicamente no en la probabilidad de aparición de las letras que no cambia, pero no cambia lo suficiente la aparición de unas letras próximas a otras, es decir las letras del texto en claro se han barajado poco. Repitiendo este proceso unas cuantas veces se llega a un texto cifrado mucho más robusto.

Data Encryption Standard (DES)

Acerca del DES hay escritos libros y libros, nos referiremos en este artículo a un pequeño resumen de sus propiedades y principios de funcionamiento. El DES nace como tal en 1977, la agencia nacional de estándares, un organismo de normalización estadounidense, lanza un concurso público para adoptar un método de cifrado. IBM presenta su método que es aceptado al ser el mejor de todos los presentados. El gigante azul estaba trabajando en un método de cifrado desde principios de los sesenta. En 1971 termina dicho proyecto, con el esotérico nombre de Lucifer. Básicamente consistía en un cifrador bloque, es decir, en la entrada se presentaban 64 bits y otros tantos a la salida ya encriptados, utilizando para ello 128 bits de clave.

Dicha agencia encontró al DES muy robusto, generando así la polémica que lo rodea desde su aparición. Los 128 bits de clave se quedaron tan solo en 56, con la consiguiente merma de robustez. La parte responsable de la potencia del sistema, las

Las técnicas de transposición consisten en alterar el orden de las letras de un texto siguiendo una regla conocida por emisor y receptor

estructuras internas de las famosas cajas S, no se hicieron públicas, permaneciendo hoy clasificadas, como si se tratara de un arma secreta. Por lo tanto sobre el DES cae la sospecha de no ser muy seguro o de que el gobierno americano posee clave maestra o de que alguien es ya capaz de descifrarlo en un corto intervalo de tiempo, etc. la 'rumurología' es muy extensa. Actualmente el DES está recomendado para uso comercial, pero no para uso de carácter estratégico, material muy sensible.

De forma resumida el DES consiste en una serie de permutaciones y sustituciones que involucran al texto en claro y a la clave, de tal manera que a la salida tenemos otros 64 bits que no se parecen en nada a los 64 bits iniciales. Es decir, que la variación en la entrada de un solo bit produce una salida completamente distinta de la anterior. Esta propiedad es conocida como el efecto avalancha. Otra propiedad del Des, es que dada la entrada todo ceros o todo unos, también es capaz de generar una salida con un número considerable de unos, y además bien repartidos.

Un ataque tipo fuerza bruta, o sea probar todas las claves posibles hasta encontrar los 64 bits originales, es actualmente inalcanzable, es decir computacionalmente seguro. Habría que probar con 2^56 claves distintas. A pesar de eso hay métodos capaces de reducir el número de claves a probar, pero no de una manera definitiva. De todas maneras es conocido que al DES le quedan unos cuantos años de vida, cuando los ordenadores sean un poco más potentes los 56 bits de clave se habrán quedado escasos.

Conclusiones

Podemos observar una constante a lo largo de la historia de la criptografía: en cuanto se descubre un método supuestamente seguro, al poco tiempo se le encuentran debilidades, que lo convierten en inseguro. Actualmente la lucha se encuentra en la potencia de los ordenadores, cuanto mas rápidos antes desencriptan un texto cifrado, y los algoritmos de cifrado que cada vez emplean más bits en sus claves, aumentando el número posible de ellas. El problema actual se ve agravado por la posibilidad de poner a trabajar en paralelo a muchos ordenadores de todo el mundo, cada uno con su porción de tarea a resolver. Solo hace falta que se lance el desafío en internet para poner a trabajar a muchos aficionados que intentan "reventar" un método de cifrado. Hace pocos años se logró descifrar el RSA, lo que obligó a aumentar el tamaño de las claves y ahora hay un intento por descifrar el DES.

Las incógnitas, guardadas bajo siete cerrojos, serían: ¿qué capacidad de desencriptado tienen las unidades de inteligencia de los distintos ejércitos? ¿Qué debilidades se han encontrado de los métodos actualmente usados y no son públicas?

Bibliografía

- "Network and Internetwork Security ", Autor William Stallings, editorial Prentice-Hall.
- "Security for computer network ", Autor Davies and Price editorial Wiley.
- "Cryptography and Secure Communication ", Autor Rhee, editorial McGraw Hill.

REDES LOCALES

Secuencias de conexión y menús en Novell Netware

Enrique Díaz Trobo

Las secuencias de conexión o login scripts y los menús de usuario son casi dos lenguajes de programación. Bien aprovechados pueden ahorrar mucho trabajo al administrador de la red. En el presente artículo describiremos ambos 'lenguajes' y veremos cómo sacarles partido, automatizando tareas que, por repetitivas y aburridas, son el tormento de muchos administradores de red.

Una de las tareas más ingratas que ha de acometer un administrador de red es el mantenimiento de los usuarios. Si no automatizamos en lo posible estas tareas, seremos unos administradores permanentemente ocupados en dar y quitar permisos, accesos a discos y aplicaciones... Los menús de usuario y las secuencias de conexión pueden ayudarnos para que, por ejemplo, cambiar una aplicación de servidor no suponga editar una a una todas las secuencias de entrada de los usuarios implicados.

Secuencias de conexión o login script

Una secuencia de conexión consiste básicamente en una serie de mandatos del sistema que se ejecutan una vez que el servidor ha validado la conexión del usuario. Las órdenes situadas en las secuencias de conexión establecen el entorno general de trabajo de los usuarios y pueden asignar unidades de red, cambiar a unidades específicas, mostrar menús e iniciar aplicaciones; también se pueden utilizar sentencias del tipo IF ... THEN para ejecutar órdenes dependiendo del día de la semana, la pertenencia del usuario a un grupo de trabajo, el tipo de estación del usuario, del sistema operativo que utilice, etc.

Cuando un usuario entra en la red se ejecuta un *login script* por omisión, que establece unos parámetros mínimos para los usuarios. Este *login script* o secuencia de conexión puede evitarse especificando un *login* personalizado. Pero veamos primero cuáles son los tipos de *secuencias de conexión* de Novell Netware.

Secuencia de conexión por omisión: se ejecuta como parte de la orden *login* y no puede modificarse, pero se anula creando una secuencia de conexión personal. Configura un entorno básico de trabajo para que podamos ejecutar órdenes del sistema.

Secuencia de conexión de contenedor: los objetos contenedores pueden tener una secuencia de conexión que se ejecuta para cada usuario que entre en ese contenedor.

Secuencia de conexión de perfil: se ejecuta en cualquier punto de la red, independientemente del contenedor, para determinados perfiles de usuario. Es muy útil para dar asignaciones y permisos especiales a operadores de consola, responsables de departamento, etc.

Secuencia de conexión personal: con este tipo de secuencia de conexión es posible establecer los parámetros de cada usuario en particular. Es modificable por el propio usuario, pero el administrador de red puede modi-

ficar este derecho *e impedir que lo haga*.

Las secuencias de conexión de contenedor y de perfil han de ser muy amplias, ya que deben cubrir necesidades generales. Para ello se debe valorar los requerimientos que tienen *todos* los usuarios del contenedor o los pertenecientes a un determinado perfil y procurar cubrir todas las necesidades.

Se podría pensar que con una secuencia de conexión que resuelva las necesidades de todos los usuarios tendríamos una red muy caótica donde todo el mundo tiene acceso a todo. Pero esto no es así gracias a que disponemos de sentencias condicionales y de una gran variedad de variables de entorno, lo que nos permitirá que un *login script* de contenedor o de perfil distinga cada tipo de usuario y le dé unos derechos y parámetros específicos. De esta forma podremos incluso llegar a obviar las secuencias de conexión personales, cuya implementación es bastante tediosa y difícil de mantener. Una práctica muy útil consiste en agrupar a los usuarios que tengan necesidades similares en la red.

Recordemos, además, que a no ser que le neguemos este derecho explícitamente, el usuario puede modificar su secuencia de conexión, dando al traste con nuestro trabajo si no es un usuario que *sepa lo que se hace*. También hay que tener en cuenta que la secuencia de conexión por omisión se ejecuta en el caso de que se ha ya creado una secuencia de conexión personal, aunque se ejecuten las secuencias de conexión del sistema y de perfil.

En cualquier caso y si por alguna razón no podemos cubrir las necesidades de todos los usuarios, podemos complementarlas con las secuencias de conexión personales, ya que éstas no son excluyentes y se ejecutan tras la secuencia de conexión de contenedor y de perfil.

¿Cuándo se ejecutan las secuencias de conexión?

Cuando un usuario entra al sistema, primero se ejecuta la secuencia de conexión del contenedor, a continuación la secuencia de conexión de perfil y finalmente la secuencia de conexión personal. Es importante tener en cuenta este orden, ya que nos permite ir desgranando niveles de especificidad. En la secuencia de conexión de contenedor pondremos unas órdenes generales válidas para todos aquellos que se conecten al contenedor; en la secuencia de conexión de perfil iremos distinguiendo ya grupos de trabajo, administradores, operadores de consola, etc. Por último, y si algún usuario tiene requerimientos especiales, la secuencia de conexión personal.

Debemos poner especial cuidado en no sobrescribir asignaciones de unidades o especificaciones de entorno en las sucesivas secuencias de conexión. Si asignamos la unidad H: para los menús de usuario en la secuencia de conexión de contenedor, debemos tener cuidado en las subsiguientes para no sobrescribirla para otro propósito.

Pese a todo, los usuarios pueden elegir si desean o no ejecutar sus secuencias de conexión. Hay una serie de parámetros de la orden *login* que cubren este propósito:

login /ns. Anula todas las secuencias de conexión. Apareceremos en el directorio *LOGIN* y no tendremos asignadas unidades de búsqueda ni variables de entorno especiales. Por ejemplo, para hacer *login* a la empresa EdtSoft y saltarse todas las secuencias de conexión (incluida la secuencia de conexión por omisión) teclearíamos:

```
LOGIN .MARIVI.EDTSOFT /NS.
```

login /s. Nos saltamos todas las secuencias de conexión, pero especificamos una secuencia de conexión alternativa. Para ello deberemos crear un archivo de texto ASCII en una unidad DOS y especificar su nombre y vía de acceso. Es muy útil si tenemos 'varias personalidades'. Si, por ejemplo pertenecemos a un grupo de trabajo digamos comercial y somos también ocasionalmente operadores de consola, podemos tener en nuestra unidad local varias secuencias de conexión y elegir la adecuada en cada momento. Por ejemplo:

```
LOGIN .MARIVI.EDTSOFT /S
      C:\CONEXIONES\OPERA.TXT
```

login /PR=objeto. Con esta opción hacemos que se ejecute la secuencia de conexión correspondiente al objeto especificado, de perfil o contenedor. Existen unas reglas a tener en cuenta para hacer uso de esta opción:

1) Sólo podemos emplearlo una vez por conexión.

2) PR ha de ser la única opción especificada para la orden *login*. Es decir, no podemos combinarlo con la opción /s por ejemplo.

3) Deberemos especificar el contexto completo del NDS, a no ser que ya nos encontremos precisamente en ese contexto.

Por ejemplo, para ejecutar la secuencia de conexión correspondiente al perfil administradores del objeto BIBISYS teclearíamos:

```
LOGIN .MARIVI.EDTSOFT /PR=.ADMINSITRA-
      DORES.BIBISYS.EDTSOFT
```

Teclear todo esto es tedioso y probablemente nos equivocaremos en múltiples ocasiones. Lo mejor será crear un archivo .BAT con esta orden.

Otra opción muy interesante es crear múltiples archivos de texto con órdenes de secuencia de conexión muy específicas, por ejemplo accesos a directorios de aplicaciones, de documentos, etc., y luego irlos 'pegando' a una secuencia de conexión mediante la sentencia *include*.

Cómo crear una secuencia de conexión

Primero vamos a dar algunas reglas:

Los comandos MAP Y ATTACH los escribiremos 'tal cual'; por ejemplo map g:=sys:apps

Para ejecutar programas DOS pondremos una almohadilla (#) delante, en este formato: #vía *de acceso/nombre del ejecutable parámetros.* Por ejemplo #h:/norton/cn.exe. Sin embargo, esta forma de ejecutar programas puede dar problemas, por lo que es más recomendable usar el sistema de menús de Netware (más adelante veremos cómo crearlos) y usar este formato para ejecutar utilidades como *pconsole.*

La longitud de línea no debe superar los 150 caracteres (en aras de la claridad, y para mantenerla, si no pasamos de 60, mejor).

Sólo se puede poner un tipo de orden en cada línea. Si necesitamos ejecutar varias veces la misma orden (por ejemplo *map*), podemos unirlas en la misma línea mediante un punto y coma:

map s1=sys:apps;
map s2=sys:contabilidad

Veamos a continuación las principales órdenes y opciones que podemos emplear en las secuencias de conexión.

• **ATTACH.** Permite conectarse a un servidor de entorno *bindery*, esto es, versiones 2.x y 3.x. Su formato es *attach servidor/usuario;clave.*

• **BREAK** *ON/OFF.* Habilita o inhabilita la posibilidad de interrumpir la secuencia de conexión mediante la pulsación de las teclas CONTROL+C. Junto con las órdenes PAUSE y WAIT, resulta muy útil para depurar y verificar secuencias de conexión.

• **CLS.** Borra la pantalla y sitúa el cursor en el extremo superior izquierdo de la pantalla.

• **COMSPEC.** Se utiliza para especificar la ubicación del COMMAND.COM del DOS. Su formato es *comspec=unidad: command.com.* Es posible asignar unidades de búsqueda a la orden COMSPEC, en ese caso sustituiremos la letra de unidad por la unidad de búsqueda.

comspec=s5:command.com.

• **CONTEXT** [contexto]. Si se usa sin el parámetro contexto muestra el contexto actual. Si se especifica un contexto, cambia el contexto actual por el especificado.

Su formato es *context* o *context contexto, como por ejemplo:*

context ou=finanzas. o=interventores.

Esta orden en combinación con IF.. THEN... es muy útil para situar a usuarios en un contexto determinado en función de su nombre, pertenencia a un grupo, etc.

• **DISPLAY.** Muestra el contenido de un archivo de texto ASCII y se usa generalmente para mostrar mensajes y avisos importantes. Su formato es *display ruta/nombre de archivo.* En realidad muestra el contenido de cualquier tipo de archivo, pero si mostramos el contenido de un fichero que use códigos de formateo, éstos también serán mostrados. Así pues deberemos crear el texto como ASCII puro.

A continuación veremos cómo, con un poquito de organización, podemos montar un completo sistema de avisos y mensajes empleando muy pocas líneas. Supongamos que nos hemos creado un directorio AVISOS y en él hemos creado varios ficheros con los nombres de los días de la semana (en este caso en inglés) y queremos mostrar cada día un aviso. Para ello sólo tendremos que incluir la orden *display sys:avisos\%day_of_week.doc.* Al ejecutarse esta orden %day_of_week será sustituido por su valor y se mostraría en pantalla el archivo correspondiente, por ejemplo friday.doc. También podemos combinarlo con instrucciones IF.. THEN y cualquier variable de entorno, de esta manera podemos crear un sistema de avisos muy completo (y también complejo). El siguiente ejemplo mostrará un fichero de texto en función de la pertenencia a un grupo.

```
if member of "interventores"
    then display sys:avisos\inter.doc
else...
```

• **DOS BREAK** ON/OFF. Habilita o inhabilita la posibilidad de que el usuario pueda detener las órdenes del DOS.

• DRIVE unidad:. Permite cambiar al usuario a una unidad, previamente asignada, distinta a la unidad asignada por omisión al usuario (generalmente la F:), por ejemplo:

drive G:

• **EXIT** *[nombre de archivo].* Detiene la ejecución de la secuencia de conexión pasando opcionalmente el control a un programa externo. El siguiente ejemplo terminaría la secuencia de conexión si el usuario pertenece a un grupo determinado y cedería el control a un menú de Netware llamado *codi.*

```
if member of "codificadores" then exit "nmenu codi"
```

• **FIRE PHASERS.** Produce un sonido de 'máquina de marcianitos'. Puede ser útil para llamar la atención sobre algún mensaje en pantalla o para las cuentas con equivalencias de seguridad de administrador del sistema. Si algún empleado conoce el *password* de alguna de esas cuentas, lo escandaloso del *login* le disuadirá de entrar como administrador del sistema. Su formato es *fire phasers n times*, donde n es el número de señales acústicas que queremos generar, por ejemplo:

```
if member of "administradores" then fire phasers 7
    times
```

• **GOTO.** Bifurca la secuencia de conexión hacia una etiqueta. Su formato es *goto etiqueta, por ejemplo:*

```
if member of "operadores" then goto operadores
....
operadores:
...
```

• **IF... THEN.** Ya hemos visto más o menos cómo funciona. Su formato completo es:

```
if condición [and] [or] [nor] then
[begin] órdenes [end]
else
órdenes
```

No vamos a explicar en *Sólo Programadores* como funciona un grupo if then else; aunque sí daremos unas notas de su particular gramática y de sus limitaciones.

La opción begin hay que emplearla siempre que queramos ejecutar más de una instrucción, y se debe cerrar con un *end*. Es decir, se ejecutarán todas las órdenes tras *begin* hasta que se encuentre *end.*

Los valores de las condiciones deben ir entre comillas dobles.

Se pueden anidar hasta diez bloques if then.

Se pueden evaluar las siguientes relaciones:

=	igual
<>	distinto
>	mayor
<	menor
>=	mayor o igual
<=	menor o igual

Además de las variables de entorno, es posible evaluar parámetros que se pasaron a la orden *login*. Podemos recuperar hasta nueve parámetros numerados del 1 al 9 (en realidad son diez, pero el parámetro 0 es la propia orden *login).* Para recuperar los parámetros que se pasaron a la orden *login*, nos referiremos a ellos como %1, %2, etc.

• **INCLUDE**. Hace que se ejecute un conjunto de órdenes contenidas en un archivo externo. Su formato es *include vía de acceso\nombre de archivo.* Esta orden es útil si nuestras secuencias de conexión son muy largas o para estructurar las secuencias de conexión 'por temas'. Por ejemplo, se pueden poner todas las órdenes de conexión relativas a proceso de textos en un archivo, las de contabilidad en otro, etc., con lo que conseguiremos una mayor estructuración y claridad de código. Su empleo tiene ventajas y peligros potenciales; debemos tener mucho cuidado sobre dónde ponemos estos archivos y quién tiene derechos sobre ellos, ya que se pueden editar con cualquier procesador de textos y ello significa que cualquiera que tenga derechos de modificación sobre estos archivos puede modificar la secuencia de entrada de un perfil de usuario o un contenedor del sistema. Como ventaja, podemos encargar su modificación a cualquier usuario cualificado sin necesidad de otorgarle permisos para modificar las secuencias de entrada. Para el resto de los usuarios sólo daremos derechos de *File Scan* y *Read.*

Si nuestra organización es realmente complicada, podemos anidar los *includes,* esto es, que el fichero al que se hace referencia en el *include* tenga a su vez otro *include*. De este modo podremos anidar hasta diez niveles.

• **MAP**. Se utiliza para asignar unidades de disco a los usuarios. Una explicación detallada de esta orden requeriría casi un artículo completo al respecto, así que nos vamos a limitar a explicar someramente alguna de las opciones de esta orden. Su formato general es *map opciones unidad:=vía de acceso.*

Map display on/off. Muestra u oculta las asignaciones que se van produciendo en la secuencia de conexión.

Map ins. Inserta una unidad de búsqueda entre las ya existentes.

Map del. Suprime la asignación de unidades.

Map root. El usuario verá la asignación como si fuera la raíz de un disco. Es decir, el usuario no puede ver los directorios superiores a la unidad de asignación, existan o no.

Map Sn. Asigna una unidad de búsqueda; donde *n* representa el número de asignación.

• **NO_DEFAULT**. Se utiliza en secuencias de conexión de perfil o contenedor para evitar que se ejecute la secuencia de entrada por omisión para el usuario. Si el usuario tuviera una secuencia de conexión personal, ésta se ejecutaría normalmente.

• **PAUSE.** Detiene la ejecución de la secuencia de conexión, que continuará cuando se pulse una tecla. Se suele poner tras una orden para la visualización de un mensaje. El siguiente ejemplo detendrá la ejecución de la secuencia de conexión tras una orden para mostrar un mensaje, de modo que el usuario pueda leer el mensaje y pulse una tecla para continuar con la secuencia de conexión.

```
display %day_of_week.txt
pause
display %login_name.txt
pause
```

• **REMARK.** La línea será tomada como un comentario e ignorada en la ejecución de la secuencia de conexión. Tiene varios formatos:

```
remark esto es un comentario
rem esto es un comentario
; esto es un comentario
esto es un comentario
```

• **SET TIME** ON/OFF. Activa o desactiva la sincronización de la hora del sistema de la estación con el servidor.

• **WRITE** "texto". Muestra el texto especificado por pantalla, y es muy útil para mostrar mensajes cortos. Se pueden utilizar variables de entorno en el texto, con la peculiaridad de que si la variable de entorno se encuentra dentro del texto entrecomillado, deberemos hacer referencia a ellas en mayúsculas. Un ejemplo nos ilustrará mejor el concepto.

La orden: write "Has entrado al sistema como"; login_name produciría la salida: Has entrado al sistema como Marivi.

Idéntico resultado conseguiríamos con la orden: write "Has entrado al sistema como %LOGIN_NAME"

En cambio: write "Has entrado al sistema como login_name" tendría el siguiente resultado: Has entrado al sistema como login_name. También disponemos de unas cuantas opciones para formatear el texto:

- \r. Retorno de carro.
- \n. Nueva línea.

• \>>. Comillas
• \7. Pitido.

Veamos un ejemplo de todo esto.

```
write "Buen servicio %LOGIN_NAME,
     \r\n te has conectado al servidor
     %FILE_SERVER"
```

Para concluir con las secuencias de conexión, en el listado 1 tenéis un ejemplo comentado. Con lo visto hasta ahora seréis capaces de entenderlo y mejorarlo. Para conseguir una mayor claridad os recomiendo seguir algún método a la hora de colocar las órdenes; un servidor, por ejemplo, coloca primero las órdenes que atienden las necesidades de *hardware*, luego las que atienden necesidades de grupo y finalmente las que atienden necesidades especiales de usuarios particulares. A la hora de mantenerlo se agradece mucho.

Los menús de usuario

Los menús de usuario de Novell son casi la continuación natural de las secuencias de entrada, de hecho podemos iniciar la ejecución de menús desde la secuencia de entrada mediante la orden *exit*. Nos pueden ser muy útiles para ayudar a realizar sus tareas a usuarios poco experimentados y para prevenirnos de los 'manazas', impidiendo que puedan ejecutar ningún programa distinto a los especificados en el menú.

A partir de la versión 3.12 de Novell Netware los menús son compilados, de modo que debemos crear primero un fichero de texto ASCII, que ha de tener la extensión SRC, y compilarlo con la orden *menumake, lo que nos dará un archivo con el mismo nombre y la extensión DAT*. Para ejecutar los menús emplearemos la orden *nmenu nombre_del_menu.*En primer lugar veremos un archivo de menú (tan sencillo como inútil) que nos servirá para tener una visión general de cómo funcionan los menús y cuál es su estructura. Más adelante veremos sus posibilidades de forma detallada. Veamos, pues, el código fuente:

```
MENU 1, MENU PRINCIPAL
     ITEM Listar archivos
          EXEC DIR *.*
     ITEM Submenú
          SHOW 2
     ITEM Salir
          EXEC EXIT
MENU 2, SUBMENÚ
     ITEM Listar archivos disco J
          EXEC J:
          EXEC DIR *.*
```

Veamos cómo funciona:

Con la primera línea definimos un primer menú (menu 1) y le damos el nombre de menú que aparecerá de cara al usuario. Con ITEM definimos un elemento del menú y el texto que aparecerá en pantalla. Finalmente las órdenes que se ejecutarán cuando el usuario elija una opción y pulse Intro sobre ella. Respecto a su funcionamiento, es muy sencillo; se ejecutarán todas las órdenes por debajo de la línea de ITEM hasta que encuentre otra línea que no sea un EXEC.

Limitaciones y órdenes

A la hora de crear menús hemos de tener en cuenta las siguientes limitaciones: el primer menú que aparezca en pantalla ha de referenciarse como MENU 1; los textos de las opciones (ITEM) no pueden superar los 40 caracteres; las líneas de código del archivo de menú no pueden superar los 80 caracteres; si se alcanza este límite, poner un signo + y continuar en la línea siguiente; no se pueden crear más de 255 pantallas de menú y la profundidad (submenús que llaman a otros submenús) no puede ser mayor de 11 niveles.

Como puede apreciarse, las limitaciones no son muy fuertes y lo normal es que ni siquiera nos acerquemos a ellas.

Las órdenes de menú son las siguientes:

• **MENU.** Como ya hemos visto, presenta el formato *MENU número, título.* El número máximo es 255.
• **ITEM.** Presenta el formato *ITEM texto {opciones}. Las opciones posibles son:*
• *Batch.* Descarga el menú de memoria, de modo que la aplicación que lanzamos dispone de más memoria para ejecutarse.
• *Chdir.* Devuelve al usuario al directorio desde el cual lanzó la aplicación.
Noecho. No se muestran las órdenes del sistema operativo para la ejecución de la aplicación.
• *Pause.* Hace una pausa al terminar la aplicación de modo que el usuario pueda leer la información de pantalla (por ejemplo un dir).
• *Show.* Muestra los comandos para ejecutar la aplicación.

Es posible asignar teclas rápidas escribiendo un acento circunflejo (^) seguido de la letra o número de opción que le queramos asignar; debe ir justo antes y pegado al texto de la opción. El ejemplo anterior, asignando teclas de opción, nos quedaría del siguiente modo (también se ha añadido la opción CHDIR en el ítem "Listar archivos disco J, de modo que vuelva al directorio previo):

```
MENU 1, MENU PRINCIPAL
     ITEM ^1Listar archivos
          EXEC DIR *.*
     ITEM ^2Submenú
          SHOW 2
     ITEM ^3Salir
          EXEC EXIT
MENU 2, SUBMENÚ
     ITEM ^1Listar archivos disco J
          EXEC J:
          EXEC DIR *.*
```

• **EXEC.** Las órdenes exec se ejecutan cuando se hace efectiva una de las opciones del menú. Su efecto es, sencillamente, teclear por nosotros la orden asociada que va detrás de la propia palabra exec. Si hay varias sentencias exec seguidas, se consideran un paquete y se ejecutan secuencialmente hasta encontrar una sentencia diferente, como item o menu.

Además de poder ejecutar cualquier orden mediante la sentencia exec, tenemos a nuestra disposición algunas formas especiales de exec:

- EXEC COMMAND. Se emplea para ejecutar archivos de órdenes del DOS (.bat)
- EXEC DOS. Abre una ventana de dos (a pantalla completa) en la que los usuarios podrán utilizar las ordenes del DOS. Para regresar al menú los usuarios deberán teclear exit.
- EXEC EXIT. Se utiliza para que los usuarios puedan abandonar el menú y volver a la línea de mandatos de su sistema operativo.
- EXEC LOGOUT. Sale del menú y desconecta al usuario del servidor. Sin la orden EXEC EXIT los usuarios no podrán salir del menú, de modo que la opción LOGOUT resulta tremendamente útil cuando, por motivos de seguridad, deseamos que el usuario que se conecta no pueda ejecutar ningún otro programa ni orden que no esté contemplada en el menú. Para hacer esto sacaremos al usuario desde la secuencia de conexión hasta un menú (mediante la orden exit en la secuencia de conexión), y una vez en el menú sólo le daremos la opción de salida mediante EXEC LOGOUT. Como el lector supondrá, no es posible salir de un menú pulsando Esc o control+c. Para salir de un menú que no tenga una opción EXIT o LOGOUT es necesario reiniciar el sistema.

- **SHOW.** Su formato es *SHOW número*. Muestra el menú que se indica en el número.

- **LOAD.** Llama a otro archivo de menú. Su formato es *LOAD nombre_de_archivo*. Se visualizará la pantalla principal del menú llamado, y los usuarios podrán volver al menú llamador pulsando la tecla Esc. Si el menú llamado tiene una opción de salida con la orden EXEC EXIT se saldrá al DOS, y no al menú que lo llamó.

- **GETx.** Es un grupo de ordenes que se utilizan para obtener entradas por teclado, como el nombre de un servidor, de usuario, etc. Existen tres formas de GET: GETO, cuya entrada es opcional. Su efecto es pegar a la sentencia EXEC lo que el usuario ingresó por teclado. GETP funciona igual que GETO, pero asigna una variable (%1, %2, etc.) a la respuesta del usuario, a la que posteriormente podremos hacer referencia en otro punto del archivo de menú. GETR pide al usuario un dato que hay que rellenar obligatoriamente. Su formato general es:

GETx texto {valor_prefijo} longitud, valor_defecto {valor_sufijo}. *Texto* es la pregunta o petición que se le presentará al usuario.

Valor_prefijo es un valor que se antepondrá pegada a la respuesta del usuario. Por ejemplo, si le estamos pidiendo un nombre de directorio, Podríamos poner como valor prefijo SYS:.

Longitud es el número máximo de caracteres que podrá introducir el usuario.

Valor_defecto es una respuesta que ya le ofrecemos al usuario, quien para aceptar este valor por omisión sólo tendrá que pulsar *Intro*.

Valor_sufijo. Es un valor que se añadirá inmediatamente al final de la respuesta del usuario. Aunque no pongamos nada en *valor_prefijo* o en *valor_sufijo*, deberemos poner las llaves abierta y cerrada.

Cuando ejecutamos una orden GETx aparece un cuadro de diálogo que

Listado 1. Ejemplo de login script

```
rem ************ Sección Hardware***************
IF %OS= 'OS2' THEN BEGIN
MAP R G:=SYS:PUBLIC%OS
;Si el sistema operativo es OS/2 se asigna la unidad G: a ;publicos2 y se verá
;como raíz del disco.
MAP INS S3:=APPS:%MACHINE%OS%OS_VERSION
;Se asigna una unidad de búsqueda hacia el directorio donde ;están las aplicaciones
;nativas de OS/2
END
ELSE MAP R G:=SYS:PUBLIC
;Fin del bloque if

rem *************Asignaciones para todos*************
map *1:=SYS:MAIL%LOGIN_NAME
;Asigno la primera unidad de red (me da igual la letra que ;tenga) al directorio personal
    del usuario.
rem **********************Grupos*****************
IF MEMBER OF "OPERADORES" EXIT NºPCONSOLE
;Si es un operador de impresión sale de la secuencia de ;conexión y ejecuta pconsole
IF MEMBER OF "COMERCIAL" INCLUDE SYS:PUBLICCOMER.INC
;Si es miembro de comercial se ejecutan las asignaciones y órdenes específicas para
;ellos, que están en el archivo comer.inc
rem ********* Necesidades Especiales*****************
IF %LOGIN_NAME="admin" THEN FIRE_PHASERS 7 TIMES
;Si alguien entra como administrador, toda la sala se enterará. Hará mucho ruido.
rem*******************Avisos y Mensajes*********************
IF DAY_OF_WEEK="friday" THEN BEGIN
;Si es viernes, dar aviso de cierre por mantenimiento.
WRITE "Os recuerdo que hoy es viernes y que por"
WRITE "mantenimientonr"
WRITE "el sistema se cierranr"
WRITE "IMPEPINABLEMENTEnr"
WRITE "a las 2 de la tarde"
PAUSE
END
```

le pedirá la información al usuario, quien tendrá que pulsar F10 después de rellenar los datos para ejecutar las órdenes asociadas. Es decir, el usuario sólo puede salir del cuadro de dialogo pulsando F10, con lo que continúa la ejecución del menú, o bien Esc, que detiene la ejecución de esa parte del menú, de modo que no llegan a ejecutarse las órdenes asociadas.

El siguiente ejemplo pedirá datos al usuario para buscar un archivo en un volumen de un servidor.

```
ITEM Buscar archivo.
    GETO Servidor: { } 11, SERV_1, {\}
    GETO Volumen: {} 11, SYS:, {}
    GETO Máscara de archivo: {} 256, *.log, {
    /S}

    EXEC NDIR
```

Cuando se pulse F10, la información introducida por el usuario se pegará inmediatamente detrás de la orden EXEC (esa es la razón del espacio previo en el primer GETO, para separar la orden NDIR de sus parámetros). Si el usuario acepta los valores por defecto, la sentencia EXEC se comportará de la siguiente manera:

```
NDIR SERV_1\SYS:*.LOG /S
```

Conseguiríamos idéntico resultado usando GETP de esta manera:

```
ITEM Buscar archivo.
    GETP Servidor: { } 11, SERV_1, {\}
    GETP Volumen: {} 11, SYS:, {}
    GETP Máscara de archivo: {} 256, *.log, {}

    EXEC NDIR %1%2%3 /S
```

El siguiente ejemplo servirá como ilustración para demostrar cómo con GETP no sólo podemos poner prefijos y sufijos, sino también tenemos la posibilidad de insertar caracteres entre las órdenes. Pedirá al usuario el nombre de *login* y la contraseña para conectarse a otro servidor.

```
ITEM Conectarse a otro servidor.
ITEM Conectarse a otro servidor
    GETP Servidor: { } 11, SERV_1, {/}
    GETP Nombre: {} 20 ,, {}
    GETP Contraseña: {} 256,, {}

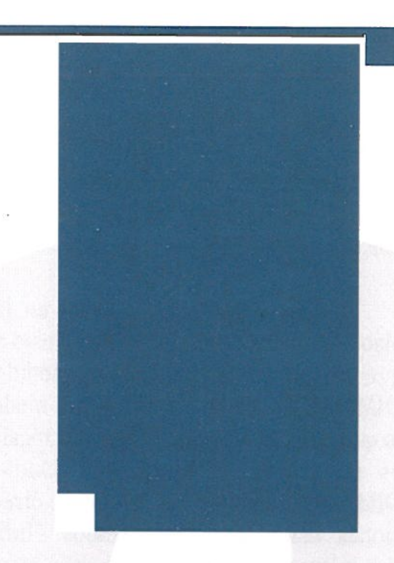
    EXEC LOGIN %1%2;%3
```

Con lo que la orden EXEC se comportaría así:

```
LOGIN SERV_1/paco;holayo
```

GETR se comporta igual que GETO, con la particularidad de que no acepta cadenas vacías.

Correo del lector



En esta sección, los lectores de SÓLO PROGRAMADORES tienen la oportunidad de hallar respuesta a los problemas que puedan tener en cualquier tema relaccionado con la programación.

SOBRE PENTIUM MMX, PENTIUM PRO Y PLACA BASE SIMM/DIMM:

Pregunta

Hola, soy un subcriptor de vuestra revista y tengo una duda muy seria aunque no es de programación sino sobre los micros PENTIUM 200MMX y los PENTIUM-PRO. Aunque os pudiera parecer muy raro, he ojeado en muchas revistas de hardware y todavía no sé qué hacer porque en las comparativas de cada revista dicen cosas distintas y espero que vosotros podáis ayudarme.

Yo quiero instalar los sistemas operativos: WINDOWS NT y LINUX, y las utilidades que voy a usar serán de programación casi todas, la multimedia no la utilizaré mucho. Lo que ahora menos me importa es el dinero porque he encontrado un sitio donde valen mas o menos lo mismo. Además me gustaría que me dijerais el nombre de una placa que creáis que es la idónea para instalar para poder además tener memoria de 72 contactos y de 168 compartidas, ambas no separadas.

Muchas gracias por vuestra atención y espero vuestra contestación pronto.

Respuesta

Pese a que hay miles de comparativas en múltiples revistas sobre las ventajas de los Pentium MMX y los Pentium Pro respecto a los Pentium normales, la verdad es que el tema a la hora de elegir uno de ellos es más simple de lo que parece.

Para simplificar, podemos decir que el Pentium Pro está diseñado y optimizado especialmente para funcionar en sistemas operativos de 32 bits, como es por ejemplo Windows NT.

Por otro lado, el Pentium MMX está diseñado para uso general en todo tipo de aplicaciones y especialmente adaptado a aplicaciones multimedia como son juegos, edición gráfica, sonido, vídeo, etc..., donde las mejoras son de hasta más de un 100%.

Descritas las dos premisas anteriores, en principio queda claro que debería decantarse por comprar un Pentium Pro, puesto que ha mencionado querer instalar Windows NT (32 bits), pero por otro lado también debe conocerse a qué área de la programación va a destinar los equipos, ya que no es lo mismo diseñar programas de gestión típicos que dedicarse a diseñar videojuegos o programas multimedia, editores gráficos u otros.

Si sabe seguro que va a dedicarse sólo a la creación de aplicaciones de algún tema muy concreto y típico, como son bases de datos, programas de contabilidad y demás, que no van a contener elementos importantes multimedia, el Pentium Pro le dará el mejor rendimiento disponible actualmente en un Micro Intel.

Por otro lado, si las aplicaciones que realiza pueden ser de tipo multimedia, interactivo, o en general, no sabe lo que tendrá que diseñar en el futuro con dichos equipos, es más recomendable adquirir un Pentium MMX, que además de ofrecer entre un 10% y un 20% más de rendimiento en aplicaciones normales no específicas para dicho modelo, le asegurará poder diseñar, en caso que se presente la necesidad, programas con capacidades MMX que podrá testear y hacer funcionar.

Como último punto, sólo recordarle que, si no tiene mucha prisa en comprar el equipo, Intel ha anunciado que presentará de forma inminente el nuevo procesador KLAMATH, llamado oficialmente Pentium II y que es un Pentium Pro con la tecnología MMX incorporada, por lo que dispondrá de las ventajas de ambos modelos en un solo chip. De hecho, la presentación del Pentium II ya ha sido realizada y estará en el mercado en breve (si no lo está ya cuando se publiquen estas líneas).

Respecto a la pregunta sobre las placas base que acepten SIMM's y DIMM's simultáneamente, la verdad es que no le debería ser demasiado difícil encontrar una.

De entrada lo más normal en las placas actuales de coste normal y bajo es que ni tan siquiera soporten módulos DIMM. Una vez se encuentra un modelo que dispone de ambos tipos de zócalo, es importante saber si los módulos DIMM usan los mismos rangos de direcciones físicas que los módulos SIMM (los mismos Bancos). Un ejemplo de este caso es que la placa disponga de 4 zócalos SIMM y 1 DIMM.

En el momento que llenásemos los 4 SIMM's con módulos, el DIMM le quedaría fuera de servicio sin posibilidad de usarlo, con lo cual será como si no los tuviera. Dada esta situación, sólo podrá usar los DIMM's que tenga instalados liberando 2 SIMM's por cada DIMM que quiera poder aprovechar. En el ejemplo, la máxima combinación permitida con ambos sería 2 SIMM's ocupados y 1 DIMM.

Pero pese a todo, existen ya modelos que permiten, como usted pide, poder usar ambos tipos de módulos simultáneamente sin tener que dejar SIMM's vacíos.

Esta capacidad la puede encontrar, según nos han comentado algunos ensambladores de ordenadores, en las placas base de la clase TRITON a partir del modelo III aunque siempre deberá asegurarse de ello examinando los manuales de la placa, ya que existen infinidad de variantes en los modelos.

A parte de todo lo dicho, si prefiere la memoria DIMM, puede adquirir una placa moderna que acepte sólo este tipo de memoria, como por ejemplo la nueva INTEL PROVIDENCE ATX/Chipset 440FX o similares, que aceptan hasta 2 procesadores (sistemas biprocesador) y son de última generación.

Jorge Figueroa

FORMATOS DE IMAGEN EN VISUAL BASIC

Pregunta

Soy un asiduo lector de su revista y este mes la sorpresa de su publicación ha sido doblemente grata. Primero por la reducción de costes, hecho que siempre es bien recibido, y en segundo lugar la inclusión de los nuevos artículos sobre criptografía.

Les felicito por la trayectoria que han mantenido hasta el momento dando a esta publicación una seriedad y profesionalidad que en estos días es difícil de encontrar en otras publicaciones nacionales e incluso en otras de ámbito internacional.

Desearía aprovechar la ocasión para presentarles una consulta:

Estoy realizando un programa en Visual Basic 4 (16bits) y utilizo controles imagen en diferentes puntos de la aplicación. El control de imagen facilitado por Microsoft sólo lee ficheros .BMP, .DIB, .WMF. Desearía saber si existe algún control imagen para leer ficheros .TIF, o .GIF (Visual Basic 4.0 - 16 bits), para poder aumentar así la calidad de gráficos de esta aplicación.

Ruego me envíen las direcciones donde puedo contactar para conseguirlos, o me aconsejen cómo solucionar este problema.

Les agradezco el tiempo que dispensaran a esta consulta.

Respuesta

Los controles de imagen de Visual Basic sólo admiten esos formatos de ficheros gráficos (.BMP, .DIB, .WMF, .WMP), porque son los únicos formatos que se pueden utilizar directamente con el GDI de Windows.

Para poder utilizar los formatos TIF y GIF en una aplicación Windows (por ejemplo la visualización) tendríamos que realizar la conversión del formato origen a uno de los formatos antes mencionados, ello requeriría la descompresión y posterior transformación entre formatos.

Para lograr dicho objetivo pueden usarse diferentes soluciones dependiendo de la utilización que deseemos de otros formatos gráficos distintos de los convencionales de Windows:

1) Si sólo deseamos mostrar imágenes estáticas en nuestros programas y los ficheros de origen están en formatos distintos del .BMP, lo más sencillo es utilizar un programa de conversión de formatos y pasarlos todos a BMP. Un buen programa que hace esto y que además es ShareWare es el archiconocido Paint Shop Pro.

2) Si lo que queremos es mostrar imágenes en diferentes formatos gráficos, pero no sabemos cuáles serán dichas imágenes, entonces podemos recurrir a librerías de terceros (VBX si estamos en Visual Basic 16 bits u OCX si estamos en 32 bits). Yo no puedo recomendarte ningún VBX en especial porque hay cientos de ellos y su elección depende mucho del dinero que desees gastar y del uso que vayas a darle. Sin embargo, si no disponemos de dinero y podemos desarrollar en 32 bits, sí puedo recomendarte un OCX, además dicho OCX es gratuito y forma parte de la distribución de Windows 95, es el OCX Imaging (desarrollado por Wang), este OCX permite la visualización e impresión de los siguientes formatos gráficos TIFF, BMP, MS FAX (AWP), JPEG y PCX.

3) En último lugar, si no quieres depender de herramientas de terceros, lo único que puedes hacer es construir tú mismo una DLL que se encargue de la descompactación de los ficheros y de la conversión de las cabeceras a formato BMP.

Esta solución es la más trabajosa pero si deseas seguirla te remito a la serie que apareció no hace mucho tiempo en esta revista dedicada a los formatos gráficos, en ella encontrarás toda la documentación que necesitarás.

Francisco López Criado

TCL 7.5, TK 4.1 FAQs sobre TCL, PGP y Solid Server 2.2

CONTENIDO DEL CD

Como cada mes analizaremos en profundidad los contenidos del CDROM. Con el del presente número esperamos llegar a una mayor cantidad de público, para ello hemos dispuesto la aplicación y los contenidos de ésta de una manera más ordenada.

Antes de nada, advertimos que los requisitos mínimos para ejecutar la aplicación del CDROM son: **8Mb de RAM, CDROM 4x o superior, 486/33 y Windows 3.1, Windows 95 ó NT.** Recomendamos también para su perfecto funcionamiento su ejecución en 256 colores. Queremos recordaros que los programas que incluimos pueden producir algunos problemas según la configuración de los equipos. Atenderemos cualquier consulta en la dirección electrónica de la revista.

Rational Rose

En el directorio **ROSE** podrá encontrar una demo de la herramienta de desarrollo visual **Rational Rose 4.0**, dicha versión demostrativa posee todas las funcionalidades del paquete original tales como generación de código e ingeniería inversa para C++. Con esta demo podrá crear o cargar modelos de cualquier tamaño pero no le permitirá salvarlos si contienen más de 30 clases, 30 módulos, 10 diagramas de transición entre estados, 3 categorías ó 3 subsistemas. En el mismo directorio podrás encontrar además la documentación sobre lenguaje **UML** en el que se basa Rational Rose; aunque los nombres de los ficheros no sean correlativos, están todos; el formato de los mismos es .PDF y para visualizarlos es necesario tener instalado el Adobe Acrobat Reader, el cual también proporcionamos en su versión más reciente dentro del directorio **\ACROBAT** (también instalable desde la aplicación del CD). Cabe recordar que esta última aplicación permite la posibilidad de insertarse como *plug-in* dentro del Netscape o Internet Explorer.

Siguiendo con herramientas de programación incluimos dos compiladores MS-DOS de 32Bits capaces de generar aplicaciones en modo protegido. El primero está localizado en el directorio **\TMT** y es un compilador shareware de Pascal, con él se encuentra un fichero **readme.doc** que describe el procedimiento de instalación del mismo. En el directorio **\DJGPP** reside nuestra segunda herramienta, un entorno de desarrollo C/C++ 32 Bits de libre distribución que se rige bajo las licencias del proyecto GNU. Este conjunto de aplicaciones (pues se trata de varios paquetes configurables para coexistir entre ellos) ha de instalarse siguiendo cautelosamente las indicaciones incluidas en el fichero **\V2\readme.1st**. Para que la configuración del programa no entrañe demasiados problemas y sea lo más fluida posible, se han adjuntado todos los FAQs que hay sobre el mismo.

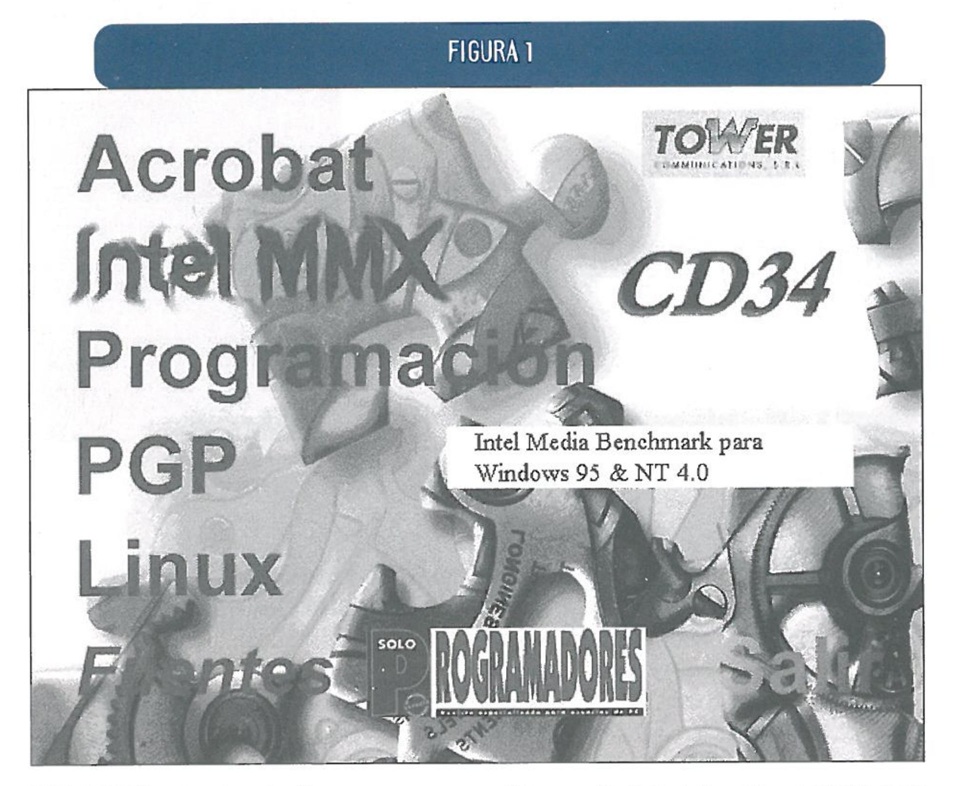
No se han incluido todos los *add-ons*, librerías y utilidades del compilador, pues se trata de un proyecto bastante grande con decenas de ficheros. En la medida de lo posible y según veamos su aceptación iremos actualizándolo con nuevas librerías, fuentes, tutoriales, etc..

En el apartado JAVA nos encontramos con **BASIC 4 JAVA**. Se trata de una herramienta de programación, pero ésta para Windows siendo compatible con Visual Basic. Capaz de generar applets JAVA y aplicaciones completamente independientes de la plataforma, el entorno de desarrollo integrado (IDE) está completamente escrito en Java, de tal manera que los usuarios ya no dependerán más de Windows como entorno de desarrollo. El fichero correspondiente a la instalación del mismo en este caso es **install.htm** y obviamente está en formato hipertexto, así que es necesario visualizarlo a través de un navegador Web.

Además, como parte del especial de este mes, incluimos un conjunto de applets que esperamos que os sean de utilidad. Podéis copiarlas desde el directorio JAVA.

Incluimos también una aplicación para medir el rendimiento de las capacidades multimedia del procesador, IN-

FIGURA 1



TEL MMX, que además sirven para procesadores MMX.

Pasamos ahora a más versiones de evaluación, esta vez de utilidades para herramientas como Visual Basic o Visual C++. En el directorio **\OLECTRA** está la revisión 5.0 del **Olectra Chart**. Con una limitación de uso de 30 días desde su instalación, se compone de un conjunto de controles ActiveX y DLLs de 16 Bits para la creación de gráficos de todo tipo. En otro directorio, en el **\INTTP** reside la versión 3.01 del **Internet Toolpack**, formado por herramientas que facilitan la integración de capacidades Internet a las aplicaciones escritas en Visual Basic.

TCL/TK

Y seguimos todavía con Windows, 32 Bits, esta vez con TCL / TK, puesto que en el directorio del mismo nombre **\TCL** se encuentran las versiones 7.5 y 4.1 del TCL y TK respectivamente. Adjuntos y por solicitud de ayuda desesperada de varios usuarios, hemos metido dos FAQs (Frequently Asked Questions), *"TCL / TK bajo Windows"* y *"TK 4+ Faq del usuario"*; esperamos así que con estos documentos los lectores encuentren menos problemas a la hora de dominar este entorno.

Como complemento a los artículos sobre criptografía, hemos incluido un especial PGP, con los mejores Shells y utilidades para el PGP y el propio PGP 2.6.3i (internacional) en versiones MS-DOS (**\PGP263I**), MS-DOS 32 Bits (**\PGP263IX**), Unix (**\UNIX**) y el código fuente del propio programa (**\PGP263IS**). Os avisamos de que la aplicación WinPGP5 tiene un error en la programación ya que al instalarse no crea los accesos (.LNK) directos de forma correcta. Para ejecutarlo deberemos crearlos nosotros.

LINUX

En el directorio **\SOLID** están todos los ficheros del servidor de bases de datos Solid Server 2.2. El programa es una versión restringida a 2 accesos concurrentes de la misma máquina desde la que se ejecuta el servidor. El paquete se compone de:

- Solid Desktop 2.2 + README
- Solid Tools (utilidades de importación / exportación de datos + README
- Solid Developer Package (libs, *.h, ejemplos) + README
- Solid Manuals (en formato HTML) + README

Los ficheros se encuentran descomprimidos en formato .tar.

Y ahora un poco de Latex, dentro del directorio del mismo nombre se encuentra un convertidor de ficheros Latex a HTML.

Y por solicitud también de nuestros lectores y esperando cumplir cada mes con la última revisión del mismo (según se vaya actualizando), está el Linux Software Map. Este documento de texto contiene un listado de todas las aplicaciones disponibles para Linux, con los datos correspondientes a la última versión de la misma, fecha, dirección del autor, dónde conseguirla y en qué consiste dicha aplicación.

En el directorio **\VBIX** tenemos un entorno de desarrollo visual para los amantes de Linux muy parecido al Visual Basic para Windows de Microsoft. Junto con él están las librerías Willow necesarias para hacerlo funcionar.

Los fuentes de la revista están en el directorio **\FUENTES** en formato comprimido y descomprimido (en sus correspondientes directorios). Y otra edición actualizada de nuestro Indice de Revistas de Sólo Programadores. Recordad que para visualizarlo debéis abrir la página **indice.htm** y desde allí empezar a navegar a través de él. Para ello hemos incorporado las últimas versiones completas corregidas del Internet Explorer (v3.02) bajo el directorio **\IE**.

Repetimos que necesitamos vuestra ayuda para seguir consiguiendo que el CD tenga los mejores contenidos, en la dirección de mail solop@towercom.es podéis enviaar las sugerencias de herramientas, shareware, demos y todo lo que os gustaría que consiguiéramos para vuestro divertimento, para el trabajo o para ayudaros en los estudios.